An extension to the multilevel logic simulator for microcomputers

de Albuquerque, Julio Cesar Lopes

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN EXTENSION TO THE
MULTILEVEL LOGIC SIMULATOR
FOR MICROCOMPUTERS

by

Julio Cesar Lopes de Albuquerque

June 1987

Thesis Advisor H. B. Rigas

NCLASSIFIED
URITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
DECLASSIFICATION/DOWNGRADING SCHEDULE	
ERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION val Postgraduate School	6b OFFICE SYMBOL (If applicable) 62
7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
ADDRESS (City, State, and ZIP Code) onterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
NAME OF FUNDING/SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)
9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

TITLE (Include Security Classification)

AN EXTENSION TO THE MULTILEVEL LOGIC SIMULATOR FOR MICROCOMPUTERS

PERSONAL AUTHOR(S)

Albuquerque, Julio Cesar Lopes de

TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
ster's Thesis	FROM _____ TO _____	1987, June	256

SUPPLEMENTARY NOTATION

COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	1. Interative Compiler 2. MULTISIM Package 3. Multilevel simulator

ABSTRACT (Continue on reverse if necessary and identify by block number)

One of the most time consuming parts of the design process is the bugging of the project. This happens when simple modifications to a rcuit require recompilation of the whole circuit.

In the CAD tool currently available for digital systems design, mpilation is a bottle neck. The VOHL system has an extremely efficient mulator phase and a reasonable but slower compilation phase.

This thesis investigates a mechanism for eliminating the need to compile the complete circuit when small changes are needed.

STRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
NAME OF RESPONSIBLE INDIVIDUAL Dr. Harriet Rigas	22b TELEPHONE (Include Area Code) (517)355-5066
	22c OFFICE SYMBOL

ORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

An Extension to the Multilevel Logic Simulator for Microcomputers

by

Julio Cesar Lopes de Albuquerque
Lieutenant Commander, Brazilian Navy
B.S., Universidade de Sao Paulo, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1987

ABSTRACT

One of the most time consuming parts of the design process is the debugging of the project. This happens when simple modifications to a circuit require recompilation of the whole circuit.

In the CAD tool currently available for digital systems design, compilation is a bottle neck. The VOHL system has an extremely efficient simulator phase and a reasonable but slower compilation phase.

This thesis investigates a mechanism for eliminating the need to recompile the complete circuit when small changes are needed.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

The project of the multilevel logic simulator began in 1981 when Ausif Mahmood, as part of his doctoral research at the Washington State University, started the construction of a VLSI logic simulator, that had been developed by Dr. H. B. Rigas. The principal idea of this development was to try to design a simulator that could have superior performance to commercial systems and be portable among a number of off-the shelf computer systems.

The system was enhanced in 1986 when Lt. J. Scott Kelly, as part of his Master of Sciences degree, made some modifications in the existing system to add facilities to the simulator.

In the initial stage of the simulator a circuit is built using the VLSI-Oriented Hardware Language, or VOHL. With this syntax the user can totally describe the circuit to be simulated. This description of the circuit is sent to the Compiler program which translates the VOHL statements into a series of data structures that allow a posterior simulation of the circuit. After the data structures are built, the Timing Wheel Simulator executes the actual simulation of the circuit.

B. THE TOOLS

1. VOHL syntax

The VOHL description language allows the user to describe the digital circuits. The complete presentation of the language was presented in Mahmood ([Ref. 1]). To allow a presentation of a brief description of the language, the circuit presented in Figure 1.1 was designed. Figure 1.2 presents the VOHL syntax for this circuit, showing all the possibilities that are available in the language for the description of a circuit.

Each circuit that we want to describe in the VOHL is called a module, and the beginning of each module is presented by the keyword MODULE, followed by the name that the user wants to give to the module. The next step in the description of the circuit is the presentation of its inputs and outputs, and these parts are defined in the syntax by the keywords INPUT and OUTPUT, respectively. After each one of these keywords the user presents the names of the variables that are the inputs and the outputs for the circuit being described.

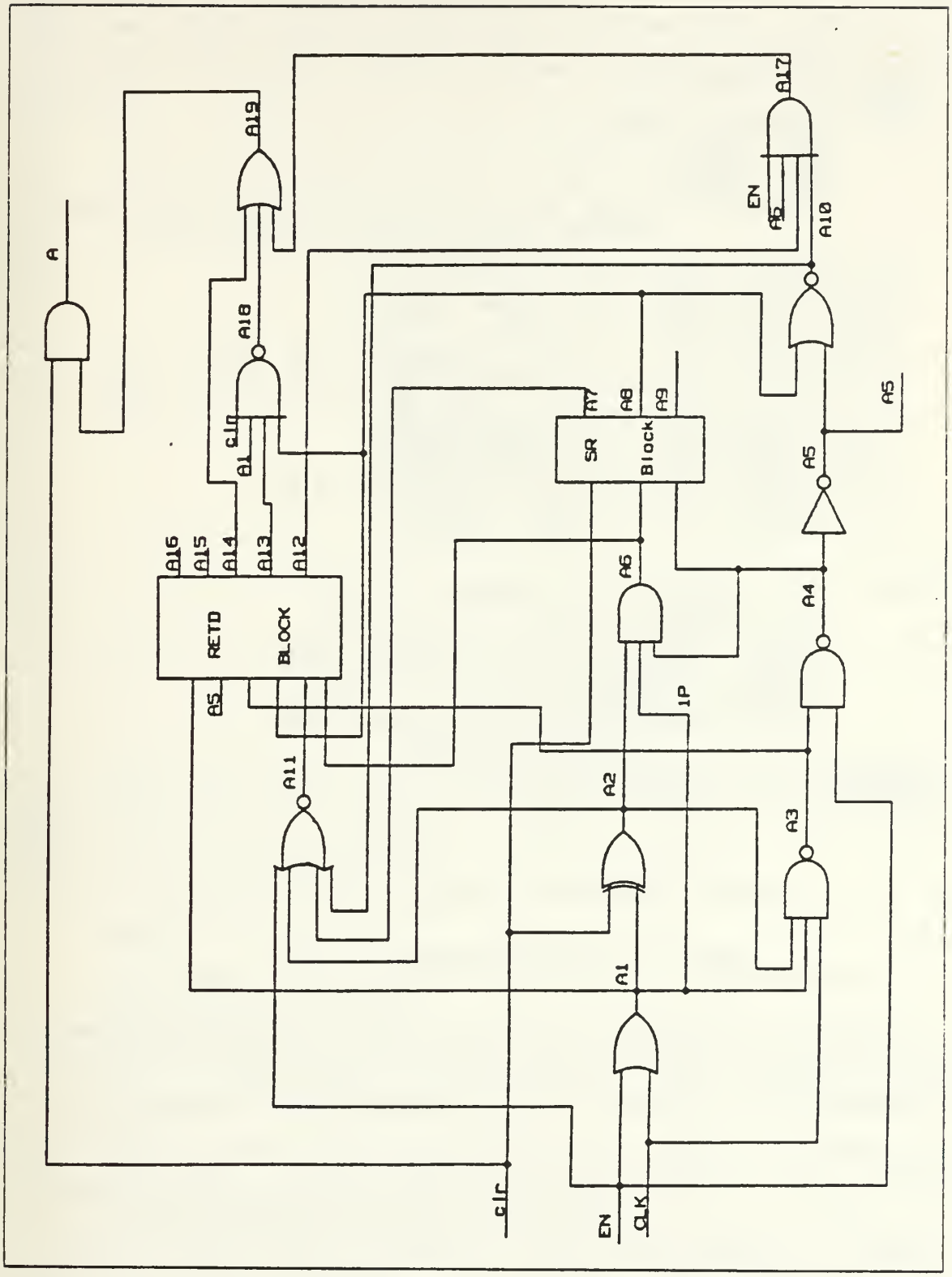


Figure 1.1 A digital circuit - schematic

```
MODULE : TEST;
INPUTS : clr, CLK, EN;
OUTPUTS : A;
TYPES : NANDTHRE : NANDT1 ;
        EXOR : EXOR1 ;
        NOR : NOR1 ;
        NAND : NAND1 ;
        INTERNALS : A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11,
                    A12, A13, A14, A15, A16, A17, A18, A19;
{
   A17 = ANDFOUR (A12, A10, A6, EN);
   A1 = OR (CLK, EN);
   USING (NOEXP,EXOR1) : A2 = EXOR (clr, A1);
   A19 = ORTHREE (A17, A18, A14);
   A5 = INVERT (A4);
   USING (NAND1) : A7, A8, A9 = SRBLOCK (clr, A6, A4);
   A12, A13, A14, A15, A16 = RETDBLO (A3, A5, A1, A8, A11, A6);
   A = AND (clr, A19);
   USING (NOEXP,NOR1) : A10 = NOR (A5, A8);
   USING (NOEXP,NAND1) : A4 = NAND (EN,A3);
   A11 = ORFOUR (A10, A7, A2, EN);
   USING (NOEXP,NANDT1) : A3 = NANDTHR (CLK,A1,A2)
   A6 = ANDTHRE (A4, A2, A1);
   A18 = NANDFOU (A1, clr, A13, A8);
}

DEFINE : AND: RISEDEL(0,0)=2, FALLDEL(0,0)=4,
               RISEDEL(1,0)=2, FALLDEL(1,0)=3;
         NOR1: FALLDEL(0,0)=3 ;
         EXOR1 : RISEDEL(1,0)=3, FALLDEL(0,0)=2;
         NANDT1 : RISEDELL(2,0)=2, FALLDEL(1,0)=3;
         ANDFOUR : FANOUT = 10 ;
         ORTHREE : OVERLOAD = 2;
         NAND1: RISEDELAY(0,0)=3, FALLDELAY(0,0)=2,
                                  RISEDELAY(1,0)=4;
INITIALIZE : A = 1, A10= 1, A3 = 0 ;
PRINTOUT : clr, CLK, EN, A5, A;
END;
```

Figure 1.2 A VOHL description of a circuit

The keyword TYPES begins the next part in the description. This part can be divided into two parts: the first part begins with the keyword INTERNALS, and lists all the variables that are restricted to the circuit being described,i.e., those that are not inputs or outputs to the circuit. The second part does not have a specified keyword to define it. In this part the user presents the names that he/she has given to those gates that have specifications that are different from the default specifications for the gates. For example, if the user wants to use one of the AND gates of the circuit with different delay values he/she might name this type of gate as AND1. This information will

appear in the description as AND : AND1, and when the compiler reads this definition it will expect a further definition of the changes for this kind of gate. All the gates that have differences with respect to the standards need to be presented in this part, with this syntax.

```
READIN 0 0 2 0
AND 2 1 2 0
OR 2 1 2 0
NAND 2 1 2 0
NOR 2 1 2 0
INVERT 1 1 2 0
EXOR 2 1 2 0
ANDTHRE 3 1 2 0
NANDTHR 3 1 2 0
SRBLOCK 3 3 2 1
RETDBLO 6 5 2 1
ANDFOUR 4 1 2 0
NANDFOU 4 1 2 0
ORTHREE 3 1 2 0
ORFOUR 4 1 2 0
```

Figure 1.3 The library of primitives

Since all the variables and modified gates that will be used in the circuit are already declared, the user can now start to describe the circuit to be simulated. The actual structure of the circuit is presented in the following scheme:

output variable = primitive name (input variables)

where:

1. output variable - one (or more) of the outputs or internals presented in the declaration part of the circuit;
2. primitive name - one of the primitives that is supported by the system in its library or an user primitive that will be described in future modules. The library of primitives that are actually supported by the system is presented in the Figure 1.3;
3. input variable - one (or more) of the variables (input, output or internals) presented in the declaration part of the circuit.

The library of primitives presented in the Figure 1.3 appears as a table, with each element having 5 different fields. A complete description of the meaning of the fields can be found in Kelly ([Ref. 2]).The fields that compose the library of primitives are:

1. The user-defined name of the primitive
2. number of inputs in the primitive
3. number of outputs in the primitive
4. type of description available for the primitive - This number shows what kind of description was made for the primitive. If the primitive has a block level description this number is 0. If the description is a structural description this number is 1 and if the description is composed by both types of descriptions this number is 2. This field is important because, depending of the type of description of the primitive, some restrictions apply to its use in both, the Compiler and the Editor programs.
5. primitive level - This field shows the level of the primitives supported by the system. If the primitives are basic gates, they are in the lowest level of the system, and consequently, the level will be 0. If the primitive is composed only by gates of level 0 it will be a level 1 primitive. If in the composition of the primitive contains at least one level 1 description, it will be a level 2 primitive, and so on.

During the description of the circuit the user also presents to the system with the gates that he/she defined in the TYPES part of the description to be used. The position of the gates are indicated by using the keyword USING, followed by the name given by the user to that specific gate. When the system reads the keyword it understands that some of the characteristics of this gate are different from the standard values, and it will expect the definition of these values in a future part of the syntax.

After the description of the circuit the user presents the control specifications for the simulation. First of all, the keyword DEFINE is used to allow the user to define gates with specifications that are different from those specified in the library. In this part the user presents the specification for the specific gates that were presented in the TYPES part of the description or for a general gate. In this part the user can specify rise delay, fall delay and fanout for a gate as well as define its functional description.

Some of the primitives presented in the library are really a collection of gates that are defined for the system and that can be treated as a single element. However, the user might need to modify some of the internal characteristics of these elements, and to do that it is necessary to expand the gate to a lower level. In the VOHL syntax expansion to a lower level is invoked by the keyword EXPAND, where the user expands a gate to a lower level and makes the necessary modifications to the gate. The Figure 1.4 shows the expansion of the RETDBLO to allow the internal gates of the circuits to have delays that are different from the standard delays.

```
MODULE : RETDBLO ;
INPUTS : CLR, D, CLK, DUM1, DUM2, DUM3 ;
OUTPUTS : Q, QC, DM1, DM2, DM3 ;
TYPES : INTERNALS : X, Y, W, Z ;
{
 X = NAND(Z, Y) ;
 Y = NANDTHRE( X,CLR, CLK) ;
 W = NANDTHRE(Y, CLK, Z) ;
 Z = NANDTHRE(W, CLR, D) ;
 Q = NAND(Y, QC) ;
 QC = (NANDTHRE(Q, CLR, W) ;
}
```

Figure 1.4 The RETDBLOck - expansion

The keyword INITIAL is used if any of the internals or outputs must be initialized to a specified value. The PRINTOUT keyword is used to show the variables that will be printed after the simulation.

If during the description of the circuit the user defines a primitive that is not one of those supported by the system this new primitive needs to be described to the system. This description is done in the same way as the original circuit and is called a submodule. The submodules will have the same syntax as the principal module, with the restrictions that the name that appears after the keyword MODULE needs to be the same name that appeared in the primitive part of the description, and that those submodules need to be described after the end of the description of the module where it was named. The Figure 1.5 presents an example of a description using submodules.

2. **Compiler**

The VOHL statements are compiled into data structures which are used by the simulator.

One of the most important structures for the system are the descriptor records. Descriptor records are explained in more detail in the Chapter 2. Briefly, they are records that describe the behavior of each element. The record is composed by various fields, each one with its corresponding function. As we can see in Figure 1.6, the record has fields for the type of primitive function, pointers for up to two inputs and fields for their values, and also fields for parameters like fall delay, rise delay, technology, fan-out and so on. The number of inputs or outputs for each gate can be

```
MODULE : ADDFOUR;
INPUTS : A0, A1, A2, A3, B0, B1, B2, B3, CI0 ;
OUTPUTS : S0, S1, S2, S3, CO3 ;
TYPES : INTERNALS : CO0, CO1, CO2 ;
{
   S0, CO0 = FULLADD (A0, B0, CI0);
   S1, CO1 = FULLADD (A1, B1, CO0);
   S2, CO2 = FULLADD (A2, B2, CO1);
   S3, CO3 = FULLADD (A3, B3, CO2);
}
DEFINE :;
INITIALIZE : CI0=0;
PRINTOUT: S3, S2, S1, S0, CO3;

 MODULE : FULLADD ;
 INPUTS : A , B , CIN ;
 OUTPUTS : S , CO ;
 TYPES : INTERNA : X , Y , Z ;
 {
 X , Y = HALFADD ( A , B );
 S , Z = HALFADD ( X , CIN );
 CO = OR ( Z , Y );
 }

 MODULE : HALFADD ;
 INPUTS : C , D ;
 OUTPUTS : T , COH ;
 TYPES : ;
 {
 T = EXOR ( C , D );
 COH = AND ( C , D );
 }
END;
```

Figure 1.5 The VOHL description with sub-modules

extended incrementally using an extension pointer. For example, the three input OR gate needs two records, one that will hold two inputs and the output and another record that holds the other input. The descriptor records are connected together by the head pointer, to form the circuit.

The Figures 1.7 and 1.8 present a simple demonstration circuit and the connections of the descriptor records for that circuit respectively. We can see that all the descriptor interconnections follow the way in what they are presented in the VOHL syntax. More detailed information about how the descriptors are built and interconnected can be found in [Ref. 3] and in [Ref. 4].

The principal function of the compiler is to create a SIMDATA file that will be used by the simulator to build the descriptor record for each descriptor. The SIMDATA file, as will be explained later on, holds all the information about each

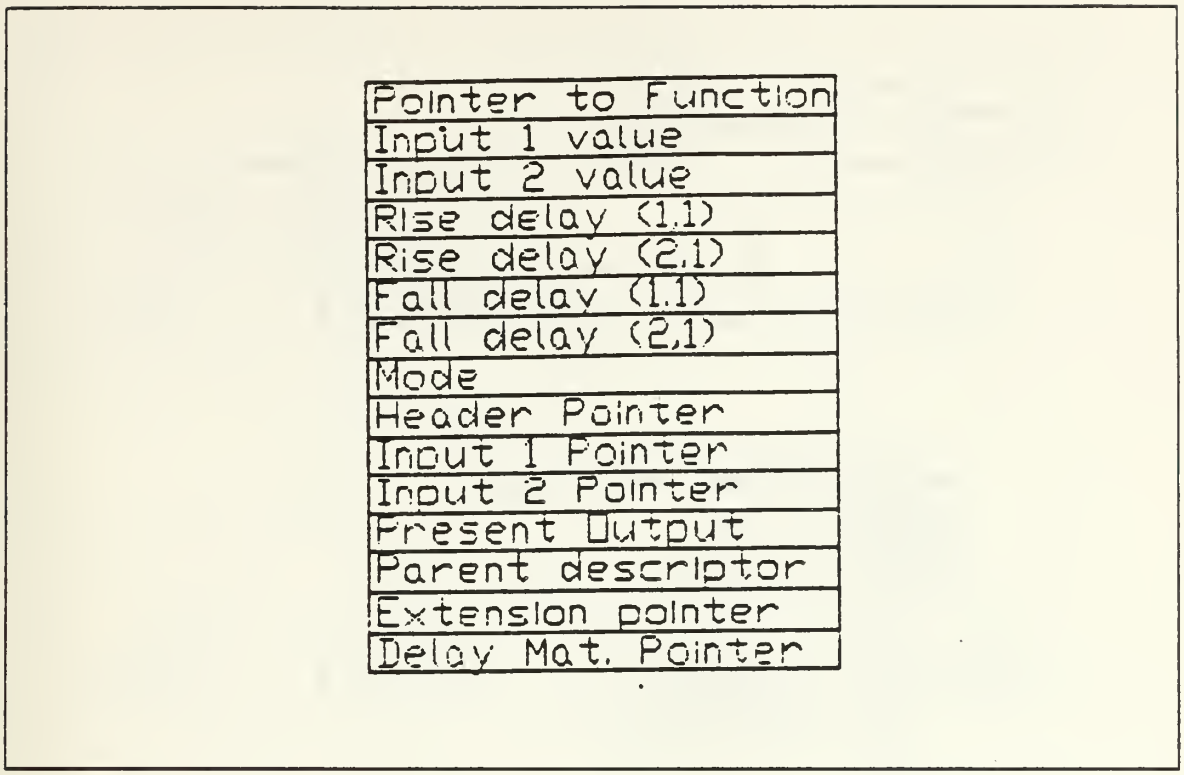
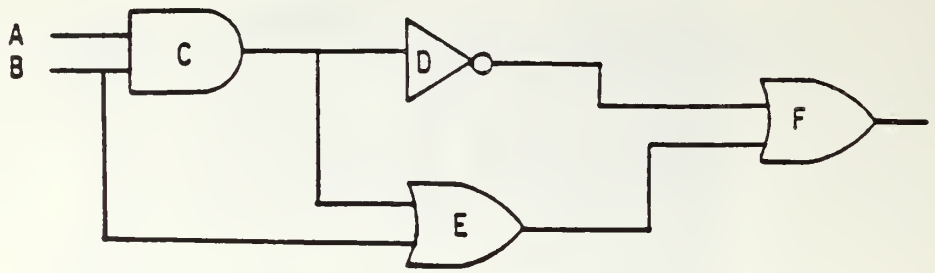


Figure 1.6 A descriptor record

descriptor, such as if it is an input, what gate is being used, delays and so on. Figure 1.9a and Figure 1.9b present the SIMDATA file created by the compiler for the circuit presented in the Figure 1.2 .

To create the SIMDATA file the compiler program first verifies whether the VOHL program describes only one module or several modules in the circuit. In the latter case, the submodules that are defined by the user are placed in an table, that is called expand table, and the structural description of these modules are moved from the user program to a temporary library for posterior usage. Also in this phase the program tries to find the keyword EXPAND and, if it is found, the module names that are found in each EXPAND line are also placed in the expand table.

Until now the system was in hierarchical form but, for simulation purposes, the system needs to be represented in a single level. To do that, the compiler looks for the expand table to verify if any expansions are requested. If the system has modules to be expanded, and if they are system primitives, the program will look for the auxiliary file STRUC, to build the single level description. If the required expansions are user defined modules the system will build the single level description using the




```
MODULE: COMPILATION_DEMO;
INPUTS: A, B;
OUTPUTS: Q;
TYPES: INTERNALS: X1, X2, X3;
   {
   X1 = AND(A, B);
   X3 = OR(X1, B);
   X2 = INVERT(X1);
   Q = OR(X2, X3);
   }
DEFINE: ;
INITIALIZE: X1=0, X2=0, X3=0;
PRINTOUT: A, B, Q;
END;
```

Figure 1.7 A demo circuit

temporary library defined at the beginning of the expansion process. Figure 1.10 shows the single level description for the circuit after all the expansions be performed.

After the single level description is ready, the program will start the construction of the SIMDATA file, that will be used in the next step by the timing wheel simulator.

3. **Simulator**

To allow an efficient simulation by only using those parts of the circuit that change state, the multilevel simulator uses the concept of the "activity stack" ([Ref. 5] and [Ref. 6]). Activity stacks are really the pointer loops that are built during the compilation of the system. When the state of a system element is modified, the pointer loop that is originated from this element receives a flag, that marks the element as

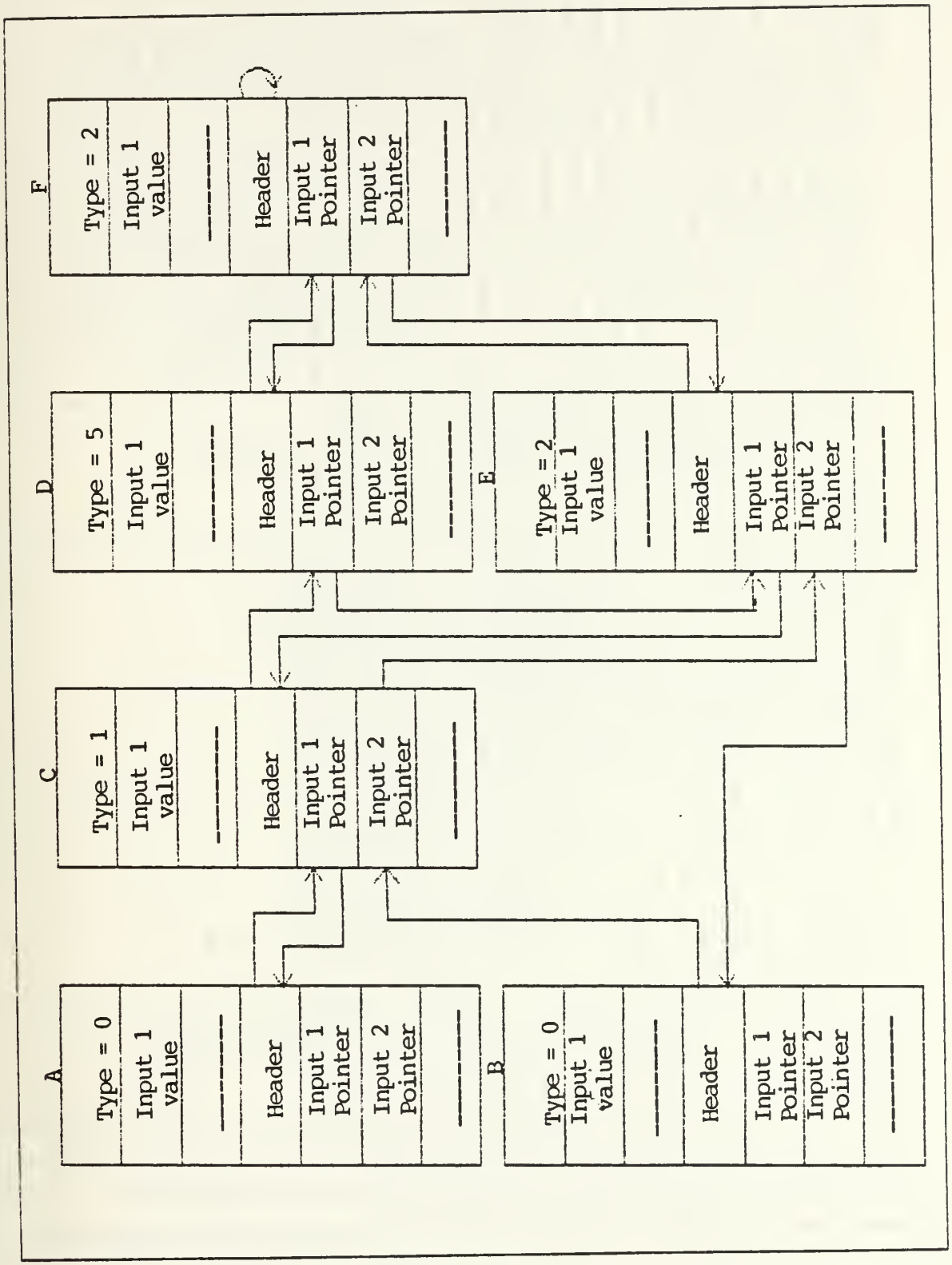


Figure 1.8 The descriptor connections for the demo circuit

```
33 0 0 1 0 0 321 1 0 2 1 33 1 0 1 1 0 218 1 1 2 1
33 2 0 1 2 0 147 1 2 2 1
33 20 11 2 15 3 15 1 15 8 20 1 15 11 0 1 20 9 0 1 20 12 0
2 13 3 13 1 13 8 20 1 13 11 1 1 20 9 1 1 20 12 1
1 20 15 23 1 23 16 20
2 9 3 9 1 9 8 23 1 9 11 0 1 23 9 0 1 23 12 0
2 2 3 2 1 2 8 23 1 2 11 1 1 23 9 1 1 23 12 1
33 4 2 2 1 3 1 1 1 8 4 1 1 11 0 1 4 9 0 1 4 12 0
2 2 3 2 1 2 8 4 1 2 11 1 1 4 9 1 1 4 12 1
33 5 6 2 0 3 0 1 0 8 5 1 0 11 0 1 5 9 0 1 5 12 0
2 4 3 4 1 4 8 5 1 4 11 1 1 5 9 1 1 5 12 1
33 22 13 2 20 3 20 1 20 8 22 1 20 11 0 1 22 9 0 1 22 12 0
2 21 3 21 1 21 8 22 1 21 11 1 1 22 9 1 1 22 12 1
1 22 15 24 1 24 16 22
2 17 3 17 1 17 8 24 1 17 11 0 1 24 9 0 1 24 12 0
33 8 5 2 7 3 7 1 7 8 8 1 7 11 0 1 8 9 0 1 8 12 0
33 10 3 2 0 3 0 1 0 8 10 1 0 11 0 1 10 9 0 1 10 12 0
2 11 3 11 1 11 8 10 1 11 11 1 1 10 9 1 1 10 12 1
33 11 3 2 10 3 10 1 10 8 11 1 10 11 0 1 11 9 0 1 11 12 0
2 9 3 9 1 9 8 11 1 9 11 1 1 11 9 1 1 11 12 1
1 15 15 16 1 16 16 15 1 16 15 17 1 17 16 15
1 17 15 18 1 18 16 15 1 18 15 19 1 19 16 15
33 15 10 2 6 3 6 1 6 8 15 1 6 11 0 1 15 9 0 1 15 12 0
2 8 3 8 1 8 8 15 1 8 11 1 1 15 9 1 1 15 12 1
2 4 3 4 1 4 8 16 1 4 11 0 1 16 9 0 1 16 12 0
2 11 3 11 1 11 8 16 1 11 11 1 1 16 9 1 1 16 12 1
2 14 3 14 1 14 8 17 1 14 11 0 1 17 9 0 1 17 12 0
2 9 3 9 1 9 8 17 1 9 11 1 1 17 9 1 1 17 12 1
33 3 1 2 0 3 0 1 0 8 3 1 0 11 0 1 3 9 0 1 3 12 0
2 22 3 22 1 22 8 3 1 22 11 1 1 3 9 1 1 3 12 1
33 13 4 2 8 3 8 1 8 8 13 1 8 11 0 1 13 9 0 1 13 12 0
2 11 3 11 1 11 8 13 1 11 11 1 1 13 9 1 1 13 12 1
33 7 3 2 2 3 2 1 2 8 7 1 2 11 0 1 7 9 0 1 7 12 0
2 6 3 6 1 6 8 7 1 6 11 1 1 7 9 1 1 7 12 1
33 14 14 2 13 3 13 1 13 8 14 1 13 11 0 1 14 9 0 1 14 12 0
2 10 3 10 1 10 8 14 1 10 11 1 1 14 9 1 1 14 12 1
1 14 15 25 1 25 16 14
2 5 3 5 1 5 8 25 1 5 11 0 1 25 9 0 1 25 12 0
2 2 3 2 1 2 8 25 1 2 11 1 1 25 9 1 1 25 12 1
33 6 8 2 1 3 1 1 1 8 6 1 1 11 0 1 6 9 0 1 6 12 0
2 4 3 4 1 4 8 6 1 4 11 1 1 6 9 1 1 6 12 1
1 6 15 26 1 26 16 6
2 5 3 5 1 5 8 26 1 5 11 0 1 26 9 0 1 26 12 0
33 9 7 2 7 3 7 1 7 8 9 1 7 11 0 1 9 9 0 1 9 12 0
2 5 3 5 1 5 8 9 1 5 11 1 1 9 9 1 1 9 12 1
1 9 15 27 1 27 16 9
2 4 3 4 1 4 8 27 1 4 11 0 1 27 9 0 1 27 12 0
33 21 12 2 4 3 4 1 4 8 21 1 4 11 0 1 21 9 0 1 21 12 0
2 0 3 0 1 0 8 21 1 0 11 1 1 21 9 1 1 21 12 1
1 21 15 28 1 28 16 21
2 16 3 16 1 16 8 28 1 16 11 0 1 28 9 0 1 28 12 0
2 11 3 11 1 11 8 28 1 11 11 1 1 28 9 1 1 28 12 1
4 20 5 2 1 5 3 1 5 4 1 5 5 1 6 20 5 2 1 5 3 1 5 4 1 5 5 1
4 4 5 2 1 5 3 1 5 4 1 5 5 1 4 5 5 2 1 5 3 1 5 4 1 5 5 1
4 22 5 2 1 5 3 1 5 4 1 5 5 1 6 22 5 2 1 5 3 1
4 8 5 2 1 5 3 1 4 10 5 2 1 5 3 1 5 4 1 5 5 1
4 11 5 2 1 5 3 1 5 4 1 5 5 1 4 15 5 2 2 5 3 2
14 0 16 0 2 17 0 2 15 0 1 16 1 2 17 1 2 18 1 2 19 1 2
15 1 2 16 2 2 17 2 2 15 2 3 16 3 2 17 3 2
6 15 5 2 2 5 3 2 5 4 2 5 5 2 14 4 16 4 2 17 4 2 18 4 2 19 4 2
15 4 5 16 5 2 17 5 2 18 5 2 19 5 2 15 5 6 16 6
2 17 6 2 18 6 2 19 6 2 15 6 7 16 7 2 17 7
```

Figure 1.9a The SIMDATA file

```
2 18 7 2 19 7 2 7  5 2 2 5 3 2 5 4 2 5 5 2
14 8 16 8 2 17 8 2 18 8 2 19 8 2 15 8 9 16 9 2 17 9 2
15 9 10 16 10 2 17 10 2 15 10 11 16 11 2 17 11 2 18 11 2 19 11 2
4 3 5 2 1 5 3 1 5 4 1 5 5 1 4 13 5 2 1 5 3 1 5 4 1 5 5 1
4 7 5 2 1 5 3 1 5 4 1 5 5 1 4 14 5 2 1 5 3 1 5 4 1 5 5 1
6 14 5 2 1 5 3 1 5 4 1 5 5 1 4 6 5 2 1 5 3 1 5 4 1 5 5 1
6 6 5 2 1 5 3 1 4 9 5 2 1 5 3 1 5 4 1 5 5 1
6 9 5 2 1 5 3 1 4 21 5 2 1 5 3 1 5 4 1 5 5 1
6 21 5 2 1 5 3 1 5 4 1 5 5 1 4 3 5 2 2 4 3 5 3 4 4 3 5 4 2
4 3 5 5 3 4 13 5 3 3 4 5 5 4 3 4 5 5 3 2 6 6 5 2 2 4 6 5 5 3
4 10 5 2 3 4 11 5 2 3 4 7 5 2 3 4 10 5 3 2 4 11 5 3 2 4 7 5 3 2
4 10 5 4 4 4 11 5 4 4 4 7 5 4 4
20 21 23 24 25 26 3 27 1 20 22 23 24 25 26 13 27 1
20 22 23 24 25 26 6 27 0 28 0 0 c
28 0 1 1 28 0 2 r 29 0 0 28 1 0 C 28 1 1 L 28 1 2 K
29 1 1 28 2 0 E 28 2 1 N 29 2 2 28 3 0 A 28 3 1 5 29 3 8
28 4 0 A 29 4 3 30 31 5 32 50
```

Figure 1.9b The SIMDATA file (continued)

```
X0 = EXOR ( A0 , B0 ) ;
Y0 = AND ( A0 , B0 ) ;
S0 = EXOR ( X0 , CI0 ) ;
Z0 = AND ( X0 , CI0 ) ;
CO0 = OR ( Z0 , Y0 ) ;
X1 = EXOR ( A1 , B1 ) ;
Y1 = AND ( A1 , B1 ) ;
S1 = EXOR ( X1 , CO0 ) ;
Z1 = AND ( X1 , CO0 ) ;
CO1 = OR ( Z1 , Y1 ) ;
X2 = EXOR ( A2 , B2 ) ;
Y2 = AND ( A2 , B2 ) ;
S2 = EXOR ( X2 , CO1 ) ;
Z2 = AND ( X2 , CO1 ) ;
CO2 = OR ( Z2 , Y2 ) ;
X3 = EXOR ( A3 , B3 ) ;
Y3 = AND ( A3 , B3 ) ;
S3 = EXOR ( X3 , CO2 ) ;
Z3 = AND ( X3 , CO2 ) ;
CO3 = OR ( Z3 , Y3 ) ;
```

Figure 1.10 A single level description

"potentially active" ([Ref. 5] and [Ref. 6]) and those stacks are scheduled to be evaluated.

Now suppose that the activity stacks that receive the flag were evaluated and both change their states (as a function of the initial state's change). In this case all the stacks that are connected to to these two stacks receive a "potentially active" flag, and

will be evaluated in the next step. If all of the flagged stacks are not modified when the initial state changes, the system will forgot them, because they will not influence the posterior function of the circuit. In the same way, the stacks that are connected to the stacks that changed their states will receive a flag to allow a posterior evaluation. This evaluation is cyclic ([Ref. 5] and [Ref. 6]), so the procedure is repeated until all the circuit's inputs have been exhausted.

The advantage of this algorithm is the speed, since only those stacks that are flagged as "potentially active" are analyzed.

C. WHY THE EDITOR

Because of the way that the simulator was implemented, the entire program must be recompiled when any modification is made to a circuit. This is not unusual in contemporary CAD tools. However, it might be possible to make small changes to the circuit without total recompilation.

In this thesis an EDITOR program is designed, to allow the user to make small modifications in a circuit without recompilation. With the EDITOR, the user will be allowed to make the following modifications in the circuit:

1. REPLACE - to replace a gate or an input to a gate by another.
2. INSERT - to insert a gate in the circuit.
3. DELETE - to delete a gate from the circuit.
4. ALTDEL - to modify a delay of one of the gates of the circuit.
5. ADDPRI - to allow the insertion of the printout of a variable in the output.
6. DELPRI - to delete the printout of a variable in the output of the circuit.
7. ALTINI - to allow the modification of an initialization value of one of the variables.
8. INSOUT - to insert an output in the circuit.
9. DELOUT - to delete an output from the circuit.
10. ALTGATE - to allow the alteration of the delays of all the gates gates of the type desired.
11. INSINP - to insert an input in the circuit.
12. INSINPG - to insert an input and a gate in the circuit.
13. DELINP - to delete an input from the circuit.
14. DELINPG - to delete an input and a gate from the circuit.

The first step in the design of the EDITOR was to understand the data structures that are generated by the original simulator. These structures will be described in the next chapter.

Once the editor is implemented the thesis investigates the conditions under which the dynamic editing is superior to recompilation.

II. ORIGINAL DATA STRUCTURES

A. AN OVERVIEW

The original multilevel simulator program creates four data structures to allow the simulation of any circuit. Those structures are two tables, one file and records. The tables, called SYMT and DESCT, are built by the compiler and contain all the information about each gate of the circuit. The file, called SIMDATA file, contains all the interconnections between the gates of the circuit to allow the timing-wheel to build the descriptor's connections for the simulation.

The Figure 2.1 presents a circuit that has almost all variations that are possible with the VOHL syntax, such as delays, initialization values, technology, and fanload that are different from the standard gates. All the tables and files that are presented as examples in this chapter are based in this circuit.

B. THE SYMT TABLE

In the original compiler program the SYMT table, that can be seen in Table 1, is a table that contains four kinds of information:

1. name of the variable
2. the descriptor number that corresponds to this variable
3. the type of gate that generates the variable, that will be the number in the PRIMITIV.DAT that corresponds to this gate (0 if the variable is an input to the circuit)
4. fanload of the variable, that is the number of gates that are connected to the output of the gate that generates the variable

This table is initiated by the compiler when the declaration part of the VOHL description of the circuit is read. The variable names are placed in the first column of the table in the order that the INPUT, OUTPUT and INTERNAL parts of the description are read, giving a descriptor number for each variable when this variable is inserted into the table. The other two columns, primitive and fanload, are written when the compiler reads the description of the circuit. As we can see, the order in which the variables appear in this table is the order in which they appear in the declaration part of the VOHL description and not in the order that they are described in the circuit.

```
MODULE : EXCKT;
INPUTS : A, A1, B;
OUTPUTS : A85;
TYPES : NANDTHRE : NANDT1 ;
        EXOR : EXOR1 ;
        NOR : NOR1 ;
        NAND : NAND1 ;
        INTERNALS : A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12,
                    A13, A14, A15, A16, A17, A18, A19, A100;
{
        A5 = INVERT (A4);
        A100 = OR (A1, B);
        A85 = AND (A, A19);
        USING (NOEXP,EXOR1) : A2 = EXOR (A, A100);
        USING (NOEXP,NOR1) : A10 = NOR (A5, A8);
        USING (NOEXP,NAND1) : A4 = NAND (B, A3);
        A19 = ORTHREE (A17, A18, A14);
        A7, A8, A9 = SRBLOCK (A, A6, A4);
        USING (NOEXP,NANDT1) : A3 = NANDTHR (A100, A1, A2);
        A6 = ANDTHRE (A4, A2, A100);
        A17 = ANDFOUR (A12, A10, A6, B);
        A11 = ORFOUR (A10, A7, A2, B);
        A18 = NANDFOU (A100, A, A13, A8);
        A12, A13, A14, A15, A16 = RETDBLO (A3, A5, A100, A8, A11, A6)
}

DEFINE : AND: RISEDEL(0,0)=2, FALLDEL(0,0)=4,
              RISEDEL(1,0)=2, FALLDEL(1,0)=3;
         NOR1: FALLDEL(0,0)=3 ;
         EXOR1 : RISEDEL(1,0)=3, FALLDEL(0,0)=2;
         NANDT1 : RISEDELL(2,0)=2, FALLDEL(1,0)=3;
         ANDFOUR : FANOUT = 10 ;
         ORTHREE : OVERLOAD = 2;
         NAND1: RISEDELAY(0,0)=3, FALLDELAY(0,0)=2,
                RISEDELAY(1,0)=4;
INITIALIZE : A85 = 1, A3= 0, A10 = 1 ;
PRINTOUT : A, A1, B, A85, A5;
END;
```

Figure 2.1 The demonstration circuit

C. THE DESCT TABLE

The DESCT table is a table that contains two types of information. The first type is the name of the gate, and the second type is the position that is occupied by this gate in the SYMT table. If we think in terms of the circuit presented in the Figure 2.1, the DESCT table for this example is the table presented in the Table 2.

This table is built when the compiler reads the description of the circuit. When the compiler reads the name of the variable, it verifies the position of this variable in

TABLE 1
THE SYMT TABLE

variable	descriptor number	primitive	fanload
A	0	0	4
A1	1	0	2
B	2	0	4
A85	3	1	0
A2	4	6	3
A3	5	8	2
A4	6	3	3
A5	7	5	2
A6	8	7	3
A7	9	9	1
A8	10	9	3
A9	11	9	0
A10	12	4	2
A11	13	14	1
A12	14	10	1
A13	15	10	1
A14	16	10	1
A15	17	10	0
A16	18	10	0
A17	19	11	1
A18	20	12	1
A19	21	13	1
A100	22	2	5

TABLE 2
THE DESCT TABLE

gate	position
INVERT	7
OR	22
AND	3
EXOR	4
NOR	12
NAND	6
ORTHREE	21
SRBLOCK	9
SRBLOCK	10
SRBLOCK	11
NANDTHR	5
ANDTHRE	8
ANDFOUR	19
ORFOUR	13
NANDFOU	20
RETDBLO	14
RETDBLO	15
RETDBLO	16
RETDBLO	17
RETDBLO	18

the SYMT table and puts the corresponding number in the second column of the table. When, in the next step, it reads the gate whose output will generate that variable, it puts the name of the gate in the first column of the table. As we can see, this table will

have its elements in the order that they appear in the description of the circuit and not in the order that they are declared.

D. THE DESCRIPTOR RECORD

The descriptor record is the data structure built by the timing-wheel simulator, following the instructions given by the SIMDATA file.

The structure of the descriptor, that can be seen in Figure 2.2, is formed by 15 fields.

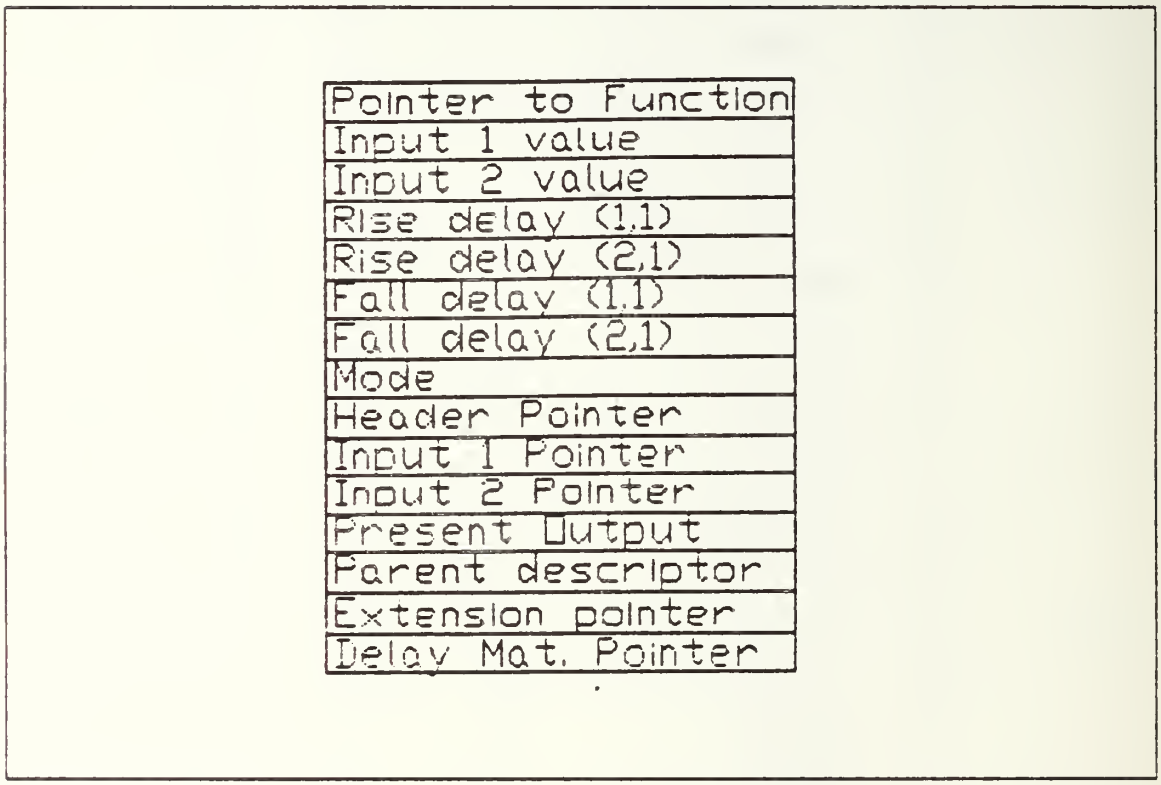


Figure 2.2 The descriptor record

The first field, the "POINTER TO FUNCTION", points to the primitive that is represented by this descriptor. This field is written when the simulator reads a descriptor in the SIMDATA file and recognizes the primitive. The next two fields are the fields "INPUT 1 VALUE" and"INPUT 2 VALUE", that will contain the values of the inputs for this descriptor. The record has also two fields called "INPUT 1 POINTER" and "INPUT 2 POINTER". These two fields are also written when the SIMDATA file is read, and they will connect the inputs to the descriptor to the

descriptor itself. The two fields that will receive the values of the inputs for the descriptor will be written during the simulation, every time that their values are computed.

The descriptor also has four fields for delays, two for rise delays and two for fall delays, with the values of the delays from each input to the output. These fields are written during the reading of the SIMDATA file or, by the default delay of the primitive (if no modification of delays is presented) or, by the modified delays to the gate that were defined in the VOHL circuit definition.

The next two fields that will be described are the fields that correspond to the output. Two fields will describe the output of the circuit. The first one is "PRESENT OUTPUT" and in this field we will have the value of the output as a function of the inputs, function and delays. The other is the "HEADER POINTER" field, and this field is a pointer to the descriptor that uses this output as one of its input.

The "MODE" field of the descriptor will be filled during the reading of the SIMDATA file, and its value will depend of the technology that is being used, the fan load for this descriptor, the fanout of the gate, etc.

As we can see, each descriptor has place for two inputs (input 1 pointer and input 2 pointer) and one output (header pointer). If we want to describe a gate with more than two inputs and/or one output one descriptor is not sufficient. In this case we will need more of these descriptors, one for each additional 2 inputs and/or 1 output. These new descriptors, called "extension descriptors", have the same fields that the original descriptor and are connected to it by the "EXTENSION POINTER" field, which is a pointer that points to the next descriptor in the chain, in a multidescriptor case. The "PARENT" field, which contains the descriptor number of the starting descriptor of this representation, is also used in this case.

Another important case for this descriptor is the gate that has more than one output, i.e., the multi-output gates. As we can see, each descriptor record has room for 4 delays, two from each input to the output, one for the value of the rise delay and one for the value of the fall delay. If the gate that is being described has more than one output we will need four more spaces for the insertion of the delay values from the inputs that are being presented in this record to each one of the new outputs. In this case the system creates one "DELAY MATRIX" extension, with the structure shown in the Figure 2.3, for each new output of the circuit. To connect the original structure to these new extensions, the "DELAY MATRIX POINTER" field is used. This field

will have a pointer that will point to any additional delay matrix extensions in the chain.

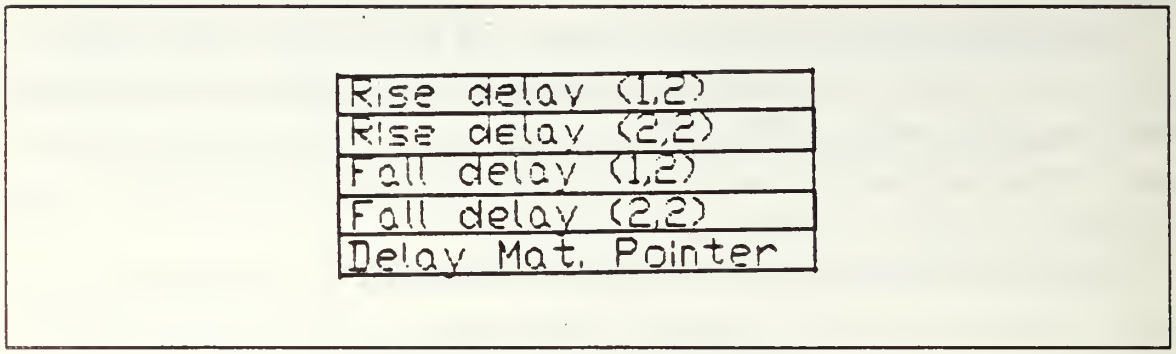


Figure 2.3 The delay matrix extension

The Figure 2.4 presents the modified descriptor structure for a multidescriptor case for a gate with N outputs and 2N inputs.

A complete presentation of the descriptor records and their interconnections can be seen in Mahmood([Ref. 7]).

E. THE SIMDATA FILE

The compiler program creates a SIMDATA file for each circuit that we want to implement. The size and the code that appears in this file will depend of the size and the specifications of the circuit that we want to simulate. Figure 2.5a and Figure 2.5b represent the SIMDATA file for the circuit that was presented in the Figure 2.1.

The SIMDATA file is composed of 5 parts that contain all the information necessary for the simulation. To create this file the compiler program generates various sets of numbers, each set having a specified meaning. The parts that compose the SIMDATA file are:

1. a part that contains all the information about the description of the circuit, in terms of what are the descriptors that appear in the circuit and how they are connected. This part is called the DESCRIPTOR PART of the file.
2. a part that contains all the default delays for each gate that is part of the circuit. This part is called DEFAULT DELAY PART.
3. a part that contains all the gates that have their delays modified by instructions of the VOHL program. This part is called MODIFIED DELAY PART.
4. a part that contains all of the initial values of variables as determined in the VOHL program. This part is called INITIALIZATION PART.

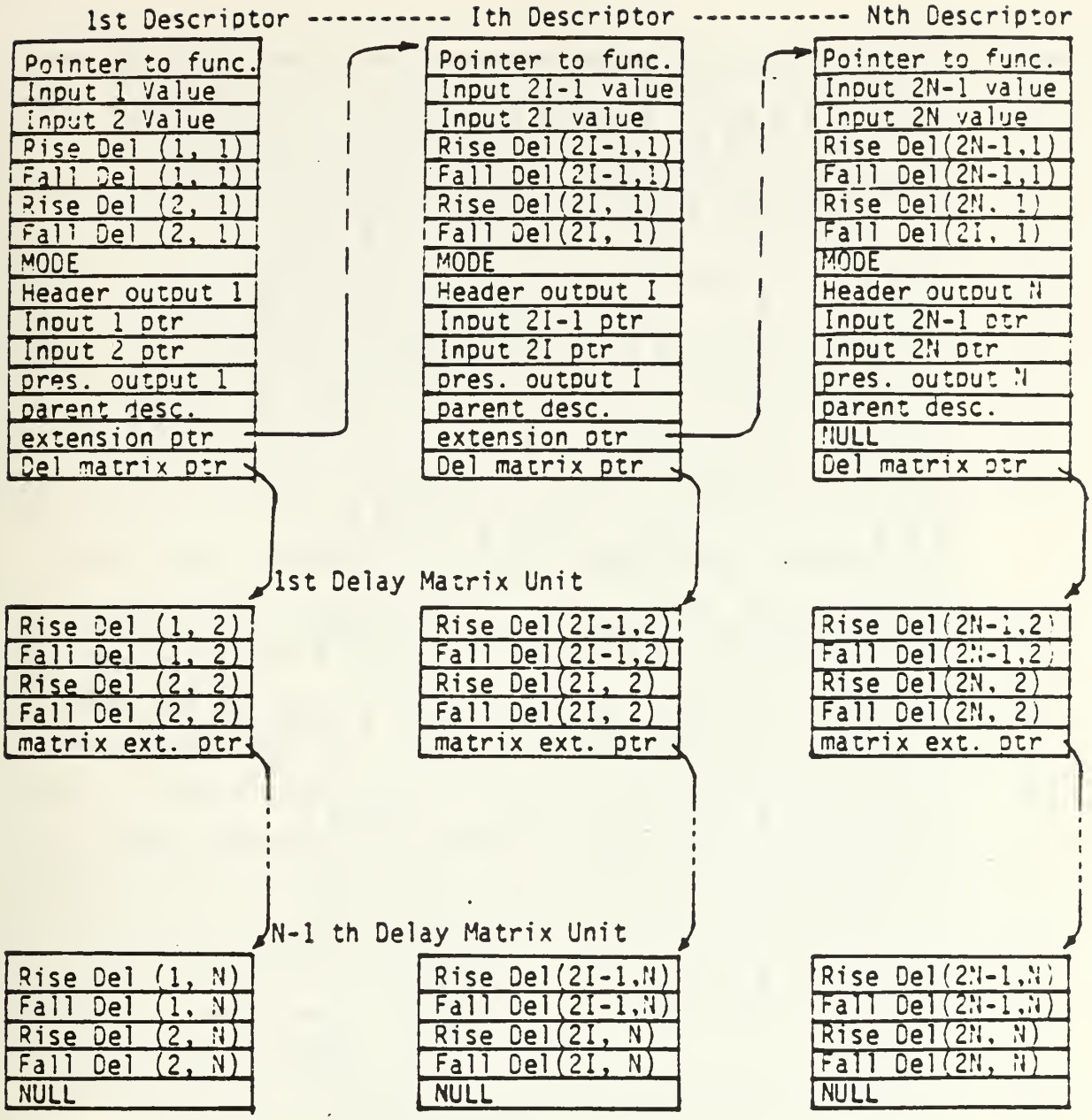


Figure 2.4 A modified descriptor structure

5. a part that contains all the variables that will be printed as output of the simulation. This part is called PRINTOUT PART.

As said before, all the parts are represented by sets of numbers. Some of these numbers really contain information about the gates or inputs that are being described and the other numbers in the set are used to inform the simulator about the next number that will appear in the file. These numbers were called "separators".

```
33 0 0 1 0 0 65 1 0 2 1 33 1 0 1 1 0 114 1 1 2 1
33 2 0 1 2 0 66 1 2 2 1
33 7 5 2 6 3 6 1 6 8 7 1 6 11 0 1 7 9 0 1 7 12 0
33 22 2 2 1 3 1 1 1 8 22 1 1 11 0 1 22 9 0 1 22 12 0
2 2 3 2 1 2 8 22 1 2 11 1 1 22 9 1 1 22 12 1
33 3 1 2 0 3 0 1 0 8 3 1 0 11 0 1 3 9 0 1 3 12 0
2 21 3 21 1 21 8 3 1 21 11 1 1 3 9 1 1 3 12 1
33 4 6 2 0 3 0 1 0 8 4 1 0 11 0 1 4 9 0 1 4 12 0
2 22 3 22 1 22 8 4 1 22 11 1 1 4 9 1 1 4 12 1
33 12 4 2 7 3 7 1 7 8 12 1 7 11 0 1 12 9 0 1 12 12 0
2 10 3 10 1 10 8 12 1 10 11 1 1 12 9 1 1 12 12 1
33 6 3 2 2 3 2 1 2 8 6 1 2 11 0 1 6 9 0 1 6 12 0
2 5 3 5 1 5 8 6 1 5 11 1 1 6 9 1 1 6 12 1
33 21 13 2 19 3 19 1 19 8 21 1 19 11 0 1 21 9 0 1 21 12 0
2 20 3 20 1 20 8 21 1 20 11 1 1 21 9 1 1 21 12 1
1 21 15 23 1 23 16 21
2 16 3 16 1 16 8 23 1 16 11 0 1 23 9 0 1 23 12 0
1 9 15 10 1 10 16 9 1 10 15 11 1 11 16 9
33 9 9 2 0 3 0 1 0 8 9 1 0 11 0 1 9 9 0 1 9 12 0
2 8 3 8 1 8 8 9 1 8 11 1 1 9 9 1 1 9 12 1
2 6 3 6 1 6 8 10 1 6 11 0 1 10 9 0 1 10 12 0
33 5 8 2 22 3 22 1 22 8 5 1 22 11 0 1 5 9 0 1 5 12 0
2 1 3 1 1 1 8 5 1 1 11 1 1 5 9 1 1 5 12 1
1 5 15 24 1 24 16 5
2 4 3 4 1 4 8 24 1 4 11 0 1 24 9 0 1 24 12 0
33 8 7 2 6 3 6 1 6 8 8 1 6 11 0 1 8 9 0 1 8 12 0
2 4 3 4 1 4 8 8 1 4 11 1 1 8 9 1 1 8 12 1
1 8 15 25 1 25 16 8
2 22 3 22 1 22 8 25 1 22 11 0 1 25 9 0 1 25 12 0
33 19 11 2 14 3 14 1 14 8 19 1 14 11 0 1 19 9 0 1 19 12 0
2 12 3 12 1 12 8 19 1 12 11 1 1 19 9 1 1 19 12 1
1 19 15 26 1 26 16 19
2 8 3 8 1 8 8 26 1 8 11 0 1 26 9 0 1 26 12 0
2 2 3 2 1 2 8 26 1 2 11 1 1 26 9 1 1 26 12 1
33 13 14 2 12 3 12 1 12 8 13 1 12 11 0 1 13 9 0 1 13 12 0
2 9 3 9 1 9 8 13 1 9 11 1 1 13 9 1 1 13 12 1
1 13 15 27 1 27 16 13
2 4 3 4 1 4 8 27 1 4 11 0 1 27 9 0 1 27 12 0
2 2 3 2 1 2 8 27 1 2 11 1 1 27 9 1 1 27 12 1
33 20 12 2 22 3 22 1 22 8 20 1 22 11 0 1 20 9 0 1 20 12 0
2 0 3 0 1 0 8 20 1 0 11 1 1 20 9 1 1 20 12 1
1 20 15 28 1 28 16 20
2 15 3 15 1 15 8 28 1 15 11 0 1 28 9 0 1 28 12 0
2 10 3 10 1 10 8 28 1 10 11 1 1 28 9 1 1 28 12 1
1 14 15 15 1 15 16 14 1 15 15 16 1 16 16 14
1 16 15 17 1 17 16 14 1 17 15 18 1 18 16 14
33 14 10 2 5 3 5 1 5 8 14 1 5 11 0 1 14 9 0 1 14 12 0
2 7 3 7 1 7 8 14 1 7 11 1 1 14 9 1 1 14 12 1
2 22 3 22 1 22 8 15 1 22 11 0 1 15 9 0 1 15 12 0
2 10 3 10 1 10 8 15 1 10 11 1 1 15 9 1 1 15 12 1
2 13 3 13 1 13 8 16 1 13 11 0 1 16 9 0 1 16 12 0
2 8 3 8 1 8 8 16 1 8 11 1 1 16 9 1 1 16 12 1
4 7 5 2 1 5 3 1 4 22 5 2 1 5 3 1 5 4 1 5 5 1
4 3 5 2 1 5 3 1 5 4 1 5 5 1 4 4 5 2 1 5 3 1 5 4 1 5 5 1
4 12 5 2 1 5 3 1 5 4 1 5 5 1 4 6 5 2 1 5 3 1 5 4 1 5 5 1
4 21 5 2 1 5 3 1 5 4 1 5 5 1 6 21 5 2 1 5 3 1
4 9 5 2 2 5 3 2 14 0 16 0 2 17 0 2 18 0 2 19 0 2
15 0 1 16 1 2 17 1 2 18 1 2 19 1 2 6 9 5 2 2 5 3 2
14 2 16 2 2 17 2 2 15 2 3 16 3 2 17 3 2
4 5 5 2 1 5 3 1 5 4 1 5 5 1 6 5 5 2 1 5 3 1
4 8 5 2 1 5 3 1 5 4 1 5 5 1 6 8 5 2 1 5 3 1
4 19 5 2 1 5 3 1 5 4 1 5 5 1 6 19 5 2 1 5 3 1 5 4 1 5 5 1
```

Figure 2.5a The SIMDATA file for the demonstration circuit

```
4 13 5 2 1 5 3 1 5 4 1 5 5 1 6 13 5 2 1 5 3 1 5 4 1 5 5 1
4 20 5 2 1 5 3 1 5 4 1 5 5 1 6 20 5 2 1 5 3 1 5 4 1 5 5 1
4 14 5 2 2 5 3 2 14 4 16 4 2 17 4 2
15 4 5 16 5 2 17 5 2 18 5 2 19 5 2 15 5 6 16 6 2 17 6 2
15 6 7 16 7 2 17 7 2 6 14 5 2 2 5 3 2 5 4 2 5 5 2
14 8 16 8 2 17 8 2 18 8 2 19 8 2 15 8 9 16 9
2 17 9 2 18 9 2 19 9 2 15 9 10 16 10 2 17 10 2 18 10
2 19 10 2 15 10 11 16 11 2 17 11 2 18 11 2 19 11 2
7  5 2 2 5 3 2 5 4 2 5 5 2 14 12 16 12
2 17 12 2 18 12 2 19 12 2
15 12 13 16 13 2 17 13 2 15 13 14 16 14 2 17 14 2
15 14 15 16 15 2 17 15 2 18 15
2 19 15 2 4 3 5 2 2 4 3 5 3 4 4 3 5 4 2
4 3 5 5 3 4 12 5 3 3 4 4 5 4 3 4 4 5 3 2 6 5 5 2 2
4 5 5 5 3 4 6 5 2 3 4 6 5 3 2 4 6 5 4 4
20 21 23 24 25 26 3 27 1 20 22 23 24 25 26 5 27 0
20 22 23 24 25 26 12 27 1
28 0 0 A 29 0 0 28 1 0 A 28 1 1 1 29 1 1
28 2 0 B 29 2 2 28 3 0 A 28 3 1 8 28 3 2 5 29 3 3 28 4 0 A
28 4 1 5 29 4 7 30 31 5 32 50
```

Figure 2.5b The SIMDATA file for the demonstration circuit(continued)

This file is built when the compiler is reading the VOHL description of the circuit. The DESCRIPTOR PART is formed when the compiler reads the structure of the circuit, with the inputs to the system in the front of the file and the other descriptors being written when the description of the circuit is read. After that the DEFAULT DELAY PART is built, using the information that was stored in the DESCT table. When the system finishes the construction of this part it reads the DEFINE part of the VOHL description and builds the MODIFIED DELAY PART of the SIMDATA file. The INITIALIZATION and the PRINTOUT parts of the file are built when the compiler reads the INITIAL and PRINT parts of the VOHL description. As we can see, the SIMDATA file is built in the same order that the circuit is read by the compiler.

For a better understanding of the SIMDATA file, the various parts from which it is formed will be discussed in the next sub-sections.

1. **The DESCRIPTOR part**

This part of SIMDATA has the function of presenting all the inputs and gates that make up the circuit that we want to simulate. Like all the other parts this part is composed of sets of numbers, each one with its own meaning. The size of each set depends of the type of gate or input that is being described. We have 3 different possibilities, that will be showed below.

a. INPUT descriptor

The descriptor that describes an input has the following format:

33 x y 1 x 0 z 1 x 2 1

As we can see it is composed of 11 numbers, with the following significance:

1. 33 - beginning of a descriptor
2. x - descriptor number, that corresponds to the number that appears in the second column of the SYMT table in the row that has the input that is being described.
3. y - primitive (same of the PRIMITIV.DAT file), that in the input case is 0.
4. z - number that represents the variable name. This number is found by the transformation of the variable name to its value in ASCII code, modified by a decimal constant of 11 if the value is the same of another variable (for example, if we have two variables called A2 and B1, the ASCII representation of both will be 115. If A2 appears first in the list of variables it will have the value 115, while B1 will have the value 126).
5. 1, 0, 1, 2, 1 - separators

In the SIMDATA file, each input will have such a descriptor. In the example shown in Figure 1 we have 3 of these descriptors.

b. Single output gate descriptor

This is the kind of descriptor that describes almost all the primitives that the system has supported until now. For each case we have the following gates:

1. with one input
   primitive 5- INVERT
2. with two inputs
   primitive 1 - AND
   primitive 2 - OR
   primitive 3 - NAND
   primitive 4 - NOR
   primitive 6 - EXOR
3. with three inputs
   primitive 7 - ANDTHRE
   primitive 8 - NANDTHR
   primitive 13 - ORTHREE
4. with four inputs
   primitive 11 - ANDFOUR
   primitive 12 - NANDFOU
   primitive 14 - ORFOUR

As shown, each descriptor has a place for 2 inputs and 1 output. As we are describing gates that have only one output, in the first two cases shown above only one descriptor record is needed to entirely describe the gate. In the cases with more than 2 inputs we will need another descriptor to put the inputs that do not fit in the record. This new descriptor record is called an "extension descriptor" and it will be connected to the original descriptor and will have the same functions as the original descriptor. The descriptor will be created in the SIMDATA file each time we finish the description of the even inputs for a gate. It will appear as a set of 8 numbers in the following way:

1 x 15 y 1 y 16 x

where:

1. x - descriptor number of the original descriptor
2. y - descriptor number of the extension descriptor

In the general case the single output gates are described in the following way:

33 x y 2 A 3 A 1 A 8 x 1 A 11 w 1 x 9 w 1 x 12 w
2 B 3 B 1 B 8 x 1 B 11 w 1 x 9 w 1 x 12 w
1 x 15 k 1 k 16 x 2 C 3 C 1 C 8 k 1 C 11 w 1 k 9 w 1 k 12 w
2 D 3 D 1 D 8 k 1 D 11 w 1 k 9 w 1 k 12 w

where:

1. 33 - beginning of a new descriptor record
2. x - descriptor number for the output (original descriptor)
3. y - primitive number(corresponds to the primitive number in the PRIMITIV.DAT file)
4. A - descriptor number of the input 1
5. B - descriptor number for the input number 2(if have)
6. C - descriptor number for the input number 3(if have)
7. D - descriptor number for the input number 4(if have)
8. w - input number (0 for inputs 1, 3, . . . and 1 for inputs 2, 4, . . .)
9. k - descriptor number for the extension descriptor(if have)

It is important to note that in the SIMDATA file will appear only in the part of the descriptor that corresponds to the number of inputs of the gate.

c. Multiple output gate descriptor

In this case, if we look at the Table 1, we can see that this type of gate has n descriptors in the table, where n is the number of outputs of the gate. In the single output case we saw that we needed to add an extension descriptor when the number of inputs was greater than 2. In the multiple output case we will need extension descriptors only when the number of inputs exceed twice the numbers of outputs, because we already have descriptors with the additional input positions available when the others descriptors have no more spaces. In this case, we can say that the "primary descriptor" is the one that appears first in the Table 1, with the others being "secondary descriptors". The interconnection of the "secondary descriptors" with the "primary descriptors" is done with the same group of numbers that does the interconnections of the extension descriptors with the original descriptor in the single output case, with the difference that in this case these interconnections appear before the "primary descriptor" , while in the single output case they appear after the original descriptor.

If we call a, b, c, . . . ,n the descriptors that correspond to the outputs of the gate and A, B, C, . . . ,P the descriptors that correspond to the inputs of the gate, the descriptor for this type of gate in the SIMDATA file will be:

1 a 15 b 1 b 16 a 1 b 15 c 1 c 16 b . . . 1 m 15 n 1 n 16 m
33 a y 2 A 3 A 1 A 8 a 1 A 11 w 1 a 9 w 1 a 12 w
2 B 3 B 1 B 8 a 1 B 11 w 1 a 9 w 1 a 12 w
2 C 3 C 1 C 8 b 1 C 11 w 1 b 9 w 1 b 12 w . . . .

where:

1. a, b, c, . . ., n - descriptor number for the outputs
2. A, B, C, . . .,P - descriptor number for the inputs
3. y - primitive number
4. w - input number (0 for the odd inputs, 1 for the even inputs)

2. The DEFAULT DELAY part

This part of SIMDATA has the function of presenting all the default delays of the gates that were used in the description of the circuit. This part is also composed of a set of numbers, each number having one value of delay from a specific input of the element to a specific output of the same element. Also in this part depending of the number of outputs of the gate that is being described, we have two different possibilities.

a. single output gates

In the general case, the default delay part of single output gates is written in the following way:

4 A 5 2 B 5 3 C 5 4 D 5 5 E

6 A 5 2 F 5 3 G . . . . where

1. A - descriptor number (same of the SYMT table)
2. B - rise delay from input 0 to output 0
3. C - fall delay from input 0 to output 0
4. D - rise delay from input 1 to output 0
5. E - fall delay from input 1 to output 0
6. F - rise delay from input 2 to output 0
7. G - fall delay from input 2 to output 0

As in the descriptor part, the only numbers that appear in this case are those that correspond to the number of inputs to the gate. The part that starts with the number 6 will appear only if the gate has more than two inputs.

b. multiple output gates

In this case, the set of numbers that describe the multi output case are more complicated than the single output case. This is due to the fact that the multi-output case does not follow a rigid formation rule, but rather depends on the number of inputs and outputs that the gate has. Basically, those descriptors are composed of the following set of numbers:

4 A 5 2 B 5 3 C 5 4 D 5 5 E

14 F 16 F G 17 F H 18 F I 19 F J

15 K L 16 L M 17 L N 18 L O 19 L P

15 L Q 16 Q R 17 Q S 18 Q T 19 Q U . . . .

6 A 5 2 a 5 3 b 5 4 c 5 5 d . . . .

7 5 2 j 5 3 k 5 4 l 5 5 m . . . .

We can divide this set of numbers into various subsets with specific functions. The first set of numbers starts in the line that begins with the number 4 and goes until the end of the line preceding the line that starts with the number 6. In this set of numbers, the delays from inputs 0 and 1 to all of the outputs of the gate are described to continue the interpretation.

1. A - descriptor number
2. B - rise delay from input 0 to the output 0
3. C - fall delay from input 0 to output 0

4. D - rise delay from input 1 to the output 0
5. E - fall delay from input 1 to the output 0

The line that starts with the number 14 describes the delays from those inputs to the second output of the circuit. The significance of the numbers in that line are:

1. F - position occupied in the matrix delay by the corresponding output. The matrix delay is the descriptor created to describe the extra delays needed to totally describe the circuit. It starts with an index 0 and increases by 1 each time that a matrix is inserted to describe extra delays.
2. G - rise delay from input 0 to the output 1
3. H - fall delay from input 0 to output 1
4. I - rise delay from input 1 to the output 1
5. J - fall delay from input 1 to the output 1

The line that starts with the number 15 describes the values of the delays from those inputs to the next output of the gate. The number of lines that begin with 15 is the number of outputs of the gate minus two, that were already described in the previous lines. The meaning of the values in those lines are:

1. K - position occupied by the previous line in the matrix delay
2. L - position occupied by this line in the matrix delay
3. M - rise delay from input 0 to the output n
4. N - fall delay from input 0 to output n
5. O - rise delay from input 1 to the output n
6. P - fall delay from input 1 to the output n

The next subset starts in the line that begins with the number 6 and finishes at the end of the line preceding the line that starts with the number 7. This subset of numbers describes the delays from inputs 2 and 3 to all the outputs of the circuit. The meaning of the numbers are the same as the first subset, except that in this subset the inputs will be 2 and 3 instead 0 and 1.

The next subset begins in the line that starts with the number 7 and finishes at the end of the line preceding the line that starts with the next 7. Each subset describes the delays from two more inputs to all the outputs of the gate, and we will have as many subsets as needed to describe all the other inputs of the circuit. The meaning of the numbers in those subsets are the same as the previous subsets, with the difference that in these subsets the descriptor number does not appear.

One important point in this case is that the values of the delays will appear in the description only if there exists a possible connection between the input and the output, i.e., if, for example there is no connection between the input 1 and the output 0, the numbers 5 4 D 5 5 E will not appear in the first line of the description.

3. **the MODIFIED DELAY part**

This part of the SIMDATA file is built only when the user defines gates with delays different of the default delays already defined. The syntax of this part will depend on the gate and the output and input that will have non standard delay values. For example, for the gate with output A2 in Figure 2.1, the syntax will be the following:

4 4 5 4 3 4 4 5 3 2

This will happen because A2 is the descriptor number 4 in the SYMT table and because the modified delays are the rise delay from the input 1 to output 0 and the fall delay from the input 0 to the output 0.

For the case of A3 in the same figure this part of the file will be:

6 5 5 2 2 4 5 5 4 3

The 6 appears in front of the descriptor number, which in this case is 5, because the delay that is being modified is the rise delay from the input 2 to the input 0. The other alteration is the fall delay from the input 1 to the input 0. This part of the SIMDATA is built in the order that the gates appear in the control part of the description of the circuit.

4. **the INITIALIZATION part**

This part of the file is built when the compiler reads the variable names that appear in the INITIAL part of the description. The code in this part appears in the following way:

20 21 23 24 25 26 a 27 b

20 22 23 24 25 26 c 27 d

20 22 23 24 25 26 e 27 f . . . .

where:

1. a, c, e, . . . - descriptor number
2. b, d, f, . . . - initial values of the descriptors a, c, e, . . . respectively

The second line of the code will be repeated until all the values that are to be initialized had being coded.

5. **the PRINTOUT part**

This part of the SIMDATA file is built when the compiler reads the PRINTOUT part of the VOHL description. This part of the file is composed of the code shown below, repeated as many times as the number of variables that we want to print.

28 a 0 A 28 a 1 B 28 a 2 C . . . 29 a x

28 (a+1) 0 I 28 (a+1) 1 J . . . 29 (a+1) y . . . .

where:

1. a, (a+1), . . . - order that the variable will appear in the printout (is the same order that is read by the compiler in the description)
2. A, B, C, . . . - characters that form the name of the variable in the position (order) a.
3. I, J, . . . - characters that form the name of the variable that appear in position (order) (a+1)
4. x, y, ... - descriptor number of each variable.

After the compiler has finished writing the code for all the variables that will be printed out, the following code is written in the file:

30 31 a 32 50

where a is the number of variables that will be printed.

For example, in the circuit shown in Figure 2.1 the code for this part will be:

28 0 0 A 29 0 0

28 1 0 A 28 1 1 1 29 1 1

28 2 0 B 29 2 2

28 3 0 A 28 3 1 8 28 3 2 5 29 3 3

28 4 0 A 28 4 1 5 29 4 7

30 31 5 32 50

This part also is responsible for finishing the SIMDATA file.

III. APPROACHES AND MODIFICATIONS FOR THE EDITOR

A. POSSIBLE APPROACHES

The multilevel simulator uses basically two data structures to allow the simulation of any circuit. The first one is the SIMDATA file, that is created by the compiler program, and the other is a set of records, called descriptor records, that is written by the timing wheel simulator. As discussed in the Chapter 2, the SIMDATA file is created while the system is compiling the circuit that we want to simulate and the descriptor records and their interconnections are done when the system is doing the simulation of the circuit.

```
MODULE : COMPILATION_DEMO ;
INPUTS : A, B ;
OUTPUTS : Q ;
TYPES : INTERNALS : X1, X2, X3 ;
{
   X1 = AND (A, B);
   X3 = OR (X1, B);
   X2 = INVERT (X1);
   Q = OR (X2, X3);
}
DEFINE :;
INITIALIZE : X1=0, X2=0, X3=0 ;
PRINTOUT: A, B, Q;
END;
```

Figure 3.1 The example circuit

For example, suppose that we want to simulate the circuit that was presented in the Figure 1.7 and is repeated in the Figure 3.1 . During the compilation of this circuit the compiler creates the file that is presented in Figure 3.2 . With the data presented in this file, the simulator program creates the necessary descriptors and makes their interconnections, presented in Figure 3.3 .

The goal of the EDITOR program is to allow the possibility of modification and simulation of a circuit, without the necessity of a new compilation of the entire circuit.

```
33 0 0 1 0 0 65 1 0 2 1 33 1 0 1 1 0 66 1 1 2 1
33 3 1 2 0 3 0 1 0 8 3 1 0 11 0 1 3 9 0 1 3 12 0
2 1 3 1 1 1 8 3 1 1 11 1 1 3 9 1 1 3 12 1
33 5 2 2 3 3 3 1 3 8 5 1 3 11 0 1 5 9 0 1 5 12 0
2 1 3 1 1 1 8 5 1 1 11 1 1 5 9 1 1 5 12 1
33 4 5 2 3 3 3 1 3 8 4 1 3 11 0 1 4 9 0 1 4 12 0
33 2 2 2 4 3 4 1 4 8 2 1 4 11 0 1 2 9 0 1 2 12 0
2 5 3 5 1 5 8 2 1 5 11 1 1 2 9 1 1 2 12 1
4 3 5 2 1 5 3 1 5 4 1 5 5 1 4 5 5 2 1 5 3 1 5 4 1 5 5 1
4 4 5 2 1 5 3 1 4 2 5 2 1 5 3 1 5 4 1 5 5 1
20 21 23 24 25 26 3 27 0 20 22 23 24 25 26 4 27 0
20 22 23 24 25 26 5 27 0 28 0 0 A 29 0 0
28 1 0 B 29 1 1 28 2 0 Q 29 2 2 30 31 3 32 50
```

Figure 3.2 The SIMDATA file for the example circuit

Suppose, for example, that we want to change the gate E of the Figure 1.7 (output X3 in the Figure 3.1) by an AND gate. The new circuit is presented in the Figure 3.4 . The SIMDATA file for this circuit is presented in Figure 3.5 and the descriptor records for the modified circuit is presented in Figure 3.6 . As we can see both structures are different from the original structures for the example circuit.

Suppose that instead of replacing one of the gates of the circuit we want to insert an AND gate between the gates E and F of the original circuit (X3 and X4 in the Figure 3.1). The new circuit is presented in Figure 3.7, with the SIMDATA file and the descriptor records for this circuit presented in Figures 3.8 and 3.9, respectively. As we can see, the structures are different from the structures presented in the previous examples.

Any modification that we want to introduce in the circuit will change both structures used for the simulation, the SIMDATA file and the descriptor records. Since the EDITOR program will skip the compilation part of the program it will need to change one of those structures. In this way, we have two possible approaches for the editor program, the first being the modification of the SIMDATA file and the second being the modification of the descriptor records and their interconnections.

If we choose the second approach some important points to be considered are:

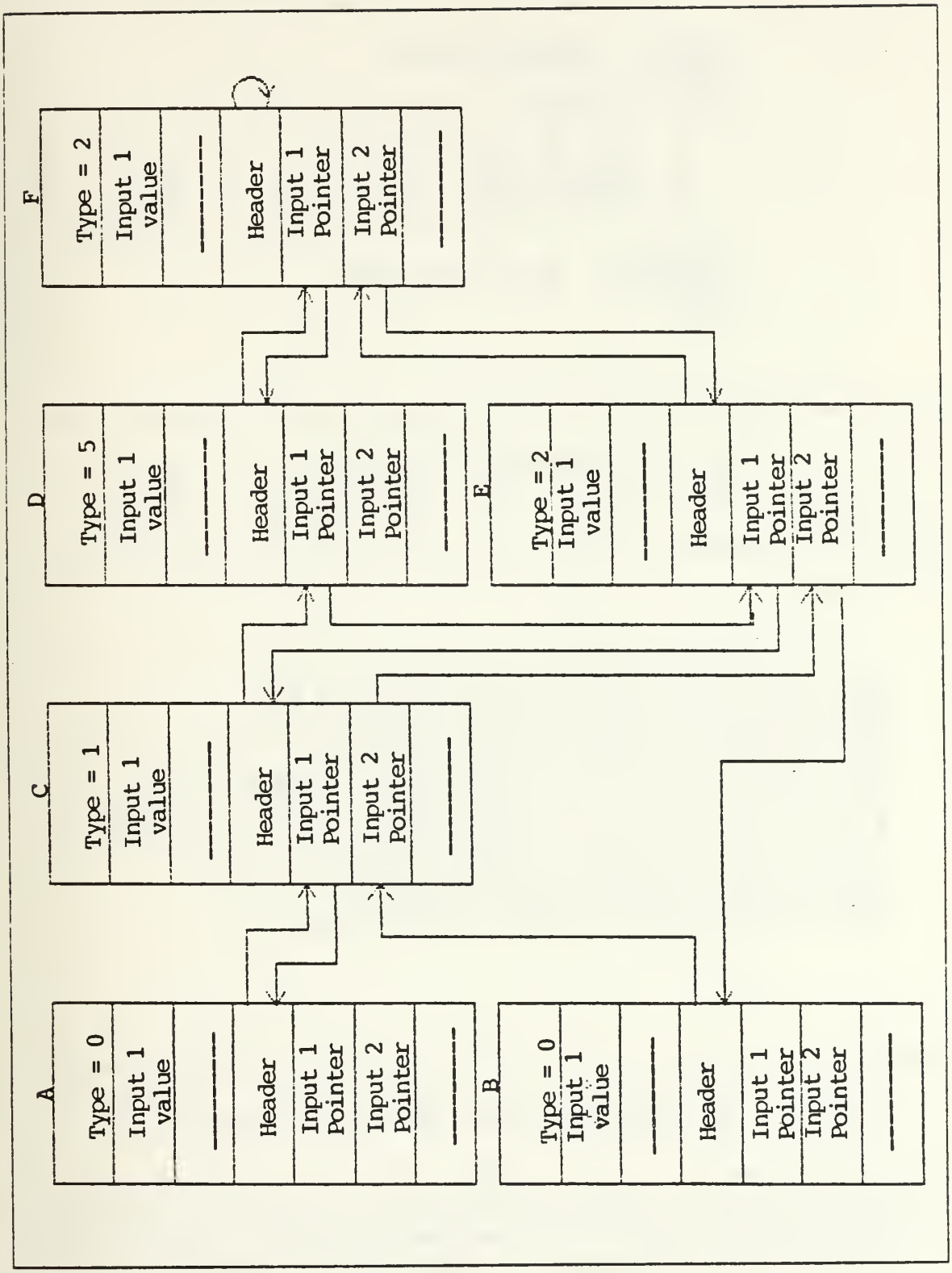


Figure 3.3 The descriptor records for the example circuit

```
MODULE : COMPILATION_DEMO ;
INPUTS : A, B ;
OUTPUTS : Q ;
TYPES : INTERNALS : X1, X2, X3 ;
{
   X1 = AND (A, B);
   X3 = AND (X1, B);
   X2 = INVERT (X1);
   Q = OR (X2, X3);
}
DEFINE :;
INITIALIZE : X1=0, X2=0, X3=0 ;
PRINTOUT: A, B, Q;
END;
```

Figure 3.4 The example circuit(modif. 1)

```
33 0 0 1 0 0 65 1 0 2 1 33 1 0 1 1 0 66 1 1 2 1
33 3 1 2 0 3 0 1 0 8 3 1 0 11 0 1 3 9 0 1 3 12 0
2 1 3 1 1 1 8 3 1 1 11 1 1 3 9 1 1 3 12 1
33 5 1 2 3 3 3 1 3 8 5 1 3 11 0 1 5 9 0 1 5 12 0
2 1 3 1 1 1 8 5 1 1 11 1 1 5 9 1 1 5 12 1
33 4 5 2 3 3 3 1 3 8 4 1 3 11 0 1 4 9 0 1 4 12 0
33 2 2 2 4 3 4 1 4 8 2 1 4 11 0 1 2 9 0 1 2 12 0
2 5 3 5 1 5 8 2 1 5 11 1 1 2 9 1 1 2 12 1
4 3 5 2 1 5 3 1 5 4 1 5 5 1 4 5 5 2 1 5 3 1 5 4 1 5 5 1
4 4 5 2 1 5 3 1 4 2 5 2 1 5 3 1 5 4 1 5 5 1
20 21 23 24 25 26 3 27 0 20 22 23 24 25 26 4 27 0
20 22 23 24 25 26 5 27 0 28 0 0 A 29 0 0
28 1 0 B 29 1 1 28 2 0 Q 29 2 2 30 31 3 32 50
```

Figure 3.5 The SIMDATA file for the example circuit (modif. 1)

1. we will have several types of possible modifications of the circuit, like replacement, insertion, or deletion of gates, inputs, and outputs, or modification of delays, change of initialization of variables, and change of printout of the circuit. To do that, the system must not only change the interconnections of the

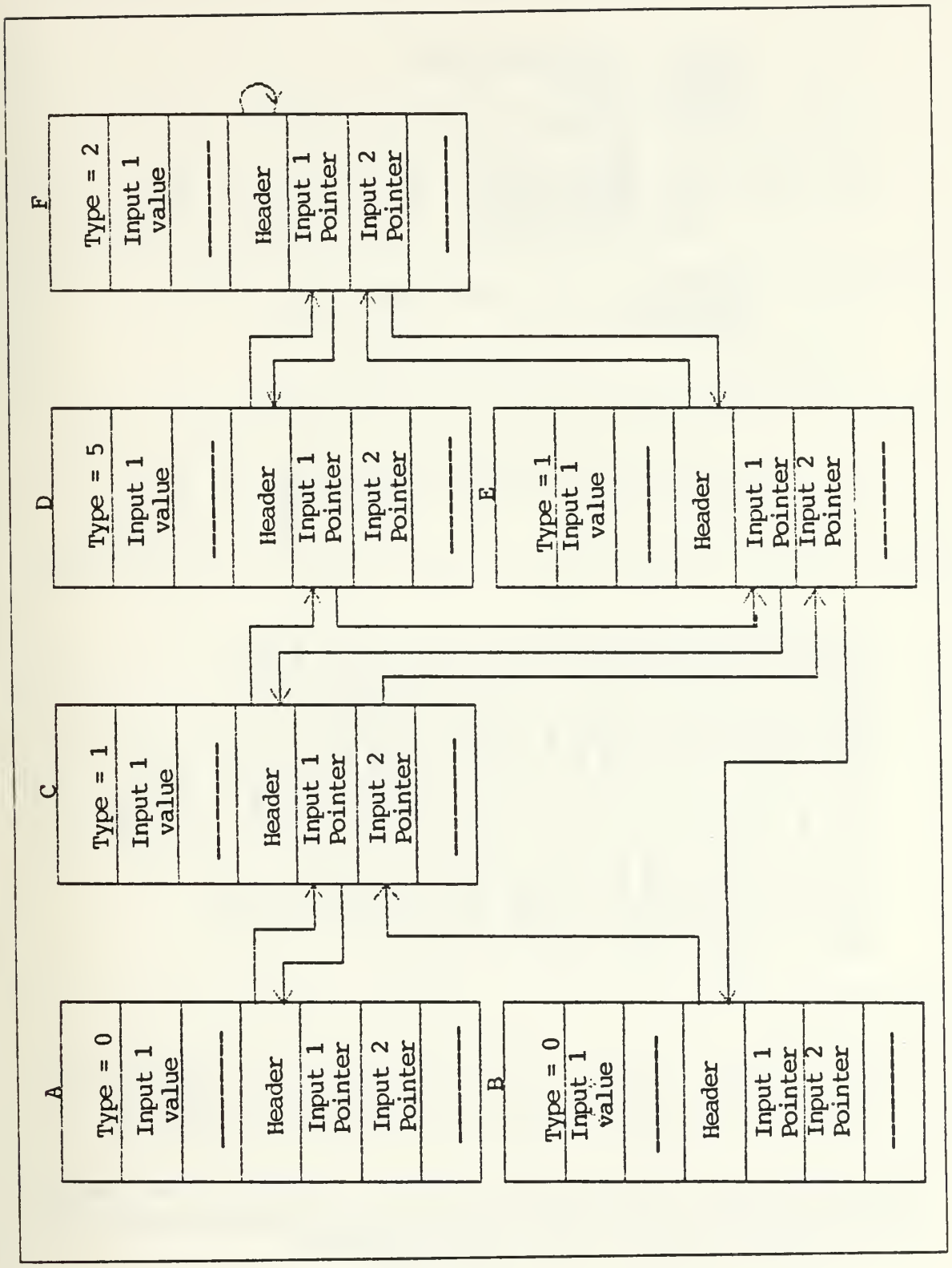


Figure 3.6 The descriptor records for the example circuit(modif. 1)

```
MODULE : COMPILATION_DEMO ;
INPUTS : A, B ;
OUTPUTS : Q ;
TYPES : INTERNALS : X1, X2, X3, X4 ;
{
   X1 = AND (A, B);
   X3 = OR (X1, B);
   X2 = INVERT (X1);
   X4 = AND (X2, X3)
   Q = OR (X2, X4);
}
DEFINE :;
INITIALIZE : X1=0, X2=0, X3=0, X4=0 ;
PRINTOUT: A, B, Q;
END;
```

Figure 3.7 The example circuit(modif. 2)

```
33 0 0 1 0 0 65 1 0 2 1 33 1 0 1 1 0 66 1 1 2 1
33 3 1 2 0 3 0 1 0 8 3 1 0 11 0 1 3 9 0 1 3 12 0
2 1 3 1 1 1 8 3 1 1 11 1 1 3 9 1 1 3 12 1
33 5 2 2 3 3 3 1 3 8 5 1 3 11 0 1 5 9 0 1 5 12 0
2 1 3 1 1 1 8 5 1 1 11 1 1 5 9 1 1 5 12 1
33 4 5 2 3 3 3 1 3 8 4 1 3 11 0 1 4 9 0 1 4 12 0
33 6 1 2 4 3 4 1 4 8 6 1 4 11 0 1 6 9 0 1 6 12 0
2 5 3 5 1 5 8 6 1 5 11 1 1 6 9 1 1 6 12 1
33 2 2 2 4 3 4 1 4 8 2 1 4 11 0 1 2 9 0 1 2 12 0
2 6 3 6 1 6 8 2 1 6 11 1 1 2 9 1 1 2 12 1
4 3 5 2 1 5 3 1 5 4 1 5 5 1 4 5 5 2 1 5 3 1 5 4 1 5 5 1
4 4 5 2 1 5 3 1 4 6 5 2 1 5 3 1 5 4 1 5 5 1
4 2 5 2 1 5 3 1 5 4 1 5 5 1
20 21 23 24 25 26 3 27 0 20 22 23 24 25 26 4 27 0
20 22 23 24 25 26 5 27 0 20 22 23 24 25 26 6 27 0
28 0 0 A 29 0 0 28 1 0 B
29 1 1 28 2 0 Q 29 2 2 30 31 3 32 50
```

Figure 3.8 The SIMDATA file for the example circuit (modif. 2)

descriptors, but in some cases also create new descriptors, delete existing descriptors, insert new delay matrices, and so on. As we can see it must be a very complicated program to allow all those possibilities.

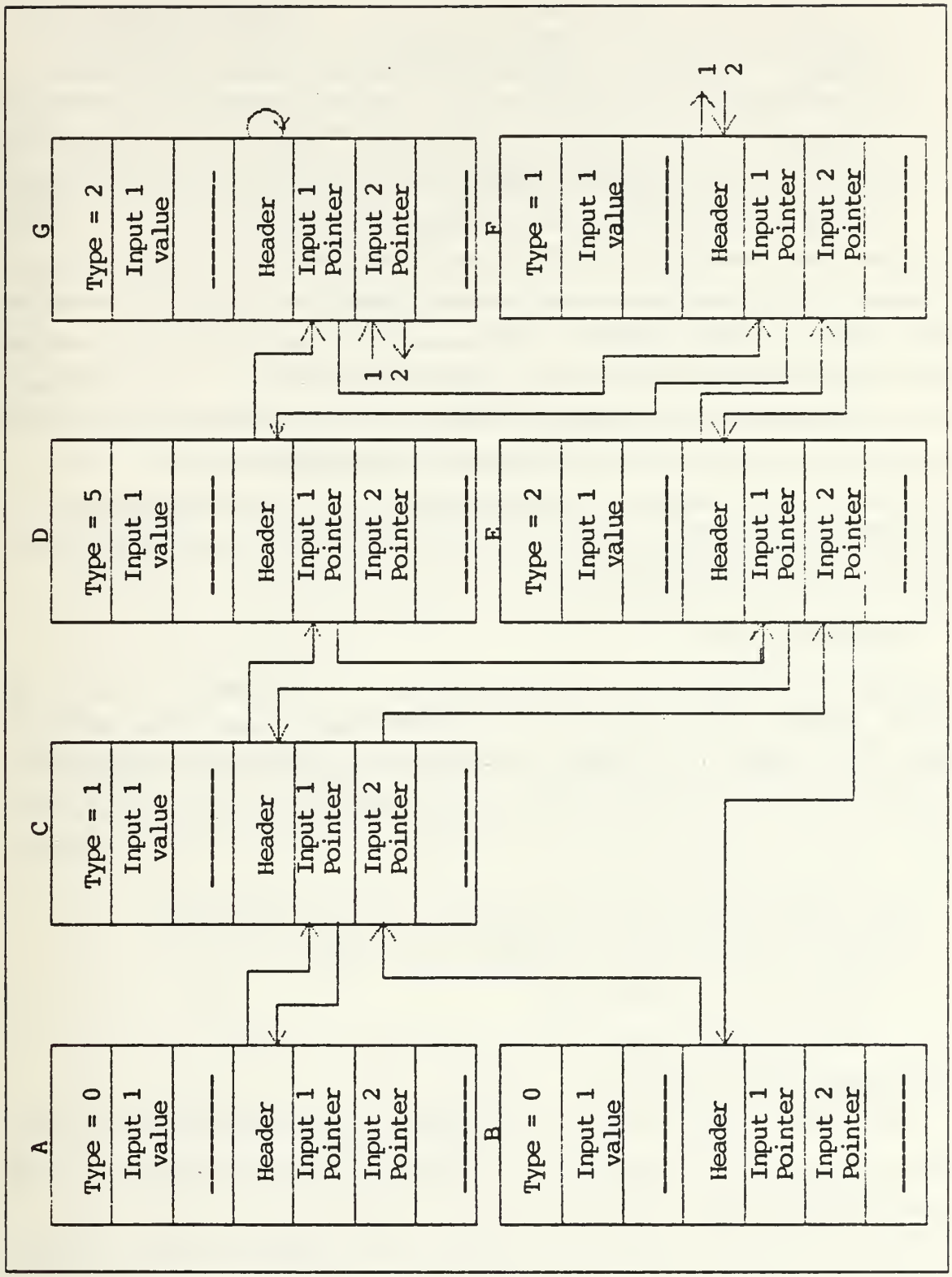


Figure 3.9 The descriptor records for the example circuit(modif. 2)

2. The timing wheel simulator reads the SIMDATA file when it starts to do a simulation. If we decide to change the descriptor records we will not have the opportunity to save the modifications that were done because the SIMDATA is not modified. The next time that we try to do the simulation of the circuit the simulator will read the SIMDATA without the modifications. As a result, any modification that was made to the circuit will be a temporary modification. If we want to do a definitive modification we must compile the circuit again, resulting in an editor that does not have a practical utilization.

If we decide to make the modification of the SIMDATA file, all the problems that were discussed above are avoided, because changing the SIMDATA file can be made a definitive change, depending of the needs of the user. Also in this case the system does not need to know if it will need to insert or delete a record, or change interconnections, because it is working with pieces of the file that are always treated in the same way, with small variations depending on what is being executed.

Following the considerations presented above we see that the best approach to the EDITOR program is the modification of the SIMDATA file and this will be the way that the editor will be designed.

B. MODIFICATIONS

The original compiler and simulator programs were not prepared to support an EDITOR program, because they were not designed with this option. The only data structure that is saved is the SIMDATA file, that contains all the information about the circuit, but this structure cannot support the editor by itself. Because of that, some modifications needed to be made in the COMPILER and TIMING WHEEL programs to allow a posterior editing of the circuit.

The modifications that were done in those programs are the following:

1. division of the SIMDATA in 5 different files.
2. modification of the SYMT table;
3. creation of the DEL table;
4. creation of the INP table

The modified CADD program and its precompilation routine (PRECOMP) can be seen in Appendix A and Appendix B. The modified timing wheel program can be seen in the Appendix C.

The next sub-sections will present in detail all the modifications that were made to the system. In each of the cases an example of the modification is presented, and all the examples will have as origin the circuit presented in the Figure 2.1.

1. **The SIMDATA file**

The different parts of the SIMDATA file are completely independent, i.e., each part of the file does not depend on others. When the modifications are made in the system only in a few cases, like the REPLACE case, will the editor need to work with the entire file. In all the other cases the modifications will be done in specific parts of the file, and it will not be necessary to read the entire SIMDATA.

Following this statement we can see that it is possible to save time during the execution of the EDITOR program if those parts are already separated. The compiler program was modified in a way to store the previous SIMDATA file in 5 different files. This modification in the program increased the compilation time in about 5%, but it saves much more time in the EDITOR program, as will be seen in the next chapters.

The 5 files that are being used now are:

1. DESCRIPTOR file - This file, that can be seen in Figure 3.10, is the copy of the descriptor part of the SIMDATA file. All the variables that appear in the description of the circuit are described in this file, with their position in the files as a function of the position where the variable was described. Its file extension will be DCF, i.e., if the circuit that is being simulated is the ALU, this file will have the name ALU.DCF.
2. DEFAULT DELAY part - This file, that can be seen in Figure 3.11, is the copy of the default delay part of the SIMDATA. As in the descriptor file, all the variables that are part of the circuit are presented in this file, with their positions in the file being a function of the position in which the variable appears in the description of the circuit. The extension that is used for this file is DDF.
3. MODIFIED DELAY file - This file, that can be seen in Figure 3.12, is the copy of the modified delay part of the SIMDATA. The only variables that appear in this file, as shown in Chapter 2, are those that, in the description of the circuit, received a delay with values different from the default values. The extension for this file is MDF.
4. INITIALIZATION file - This file, that can be seen in Figure 3.13, is the copy of the initialization part of the SIMDATA. Also in this case the only variables that appear in this file are those that were initialized in the INITIAL part of the description of the circuit. The extension for this file is INT.
5. PRINTOUT file - This file, that can be seen in Figure 3.14, is the copy of the printout part of the SIMDATA. The only variables that appear in this file were requested for printout in the description of the circuit. The extension for this file is PTT.

```
33 0 0 1 0 0 65 1 0 2 1 33 1 0 1 1 0 114 1 1 2 1
33 2 0 1 2 0 66 1 2 2 1
33 7 5 2 6 3 6 1 6 8 7 1 6 11 0 1 7 9 0 1 7 12 0
33 22 2 2 1 3 1 1 1 8 22 1 1 11 0 1 22 9 0 1 22 12 0
2 2 3 2 1 2 8 22 1 2 11 1 1 22 9 1 1 22 12 1
33 3 1 2 0 3 0 1 0 8 3 1 0 11 0 1 3 9 0 1 3 12 0
2 21 3 21 1 21 8 3 1 21 11 1 1 3 9 1 1 3 12 1
33 4 6 2 0 3 0 1 0 8 4 1 0 11 0 1 4 9 0 1 4 12 0
2 22 3 22 1 22 8 4 1 22 11 1 1 4 9 1 1 4 12 1
33 12 4 2 7 3 7 1 7 8 12 1 7 11 0 1 12 9 0 1 12 12 0
2 10 3 10 1 10 8 12 1 10 11 1 1 12 9 1 1 12 12 1
33 6 3 2 2 3 2 1 2 8 6 1 2 11 0 1 6 9 0 1 6 12 0
2 5 3 5 1 5 8 6 1 5 11 1 1 6 9 1 1 6 12 1
33 21 13 2 19 3 19 1 19 8 21 1 19 11 0 1 21 9 0 1 21 12 0
2 20 3 20 1 20 8 21 1 20 11 1 1 21 9 1 1 21 12 1
1 21 15 23 1 23 16 21
2 16 3 16 1 16 8 23 1 16 11 0 1 23 9 0 1 23 12 0
1 9 15 10 1 10 16 9 1 10 15 11 1 11 16 9
33 9 9 2 0 3 0 1 0 8 9 1 0 11 0 1 9 9 0 1 9 12 0
2 8 3 8 1 8 8 9 1 8 11 1 1 9 9 1 1 9 12 1
2 6 3 6 1 6 8 10 1 6 11 0 1 10 9 0 1 10 12 0
33 5 8 2 22 3 22 1 22 8 5 1 22 11 0 1 5 9 0 1 5 12 0
2 1 3 1 1 1 8 5 1 1 11 1 1 5 9 1 1 5 12 1
1 5 15 24 1 24 16 5
2 4 3 4 1 4 8 24 1 4 11 0 1 24 9 0 1 24 12 0
33 8 7 2 6 3 6 1 6 8 8 1 6 11 0 1 8 9 0 1 8 12 0
2 4 3 4 1 4 8 8 1 4 11 1 1 8 9 1 1 8 12 1
1 8 15 25 1 25 16 8
2 22 3 22 1 22 8 25 1 22 11 0 1 25 9 0 1 25 12 0
33 19 11 2 14 3 14 1 14 8 19 1 14 11 0 1 19 9 0 1 19 12 0
2 12 3 12 1 12 8 19 1 12 11 1 1 19 9 1 1 19 12 1
1 19 15 26 1 26 16 19
2 8 3 8 1 8 8 26 1 8 11 0 1 26 9 0 1 26 12 0
2 2 3 2 1 2 8 26 1 2 11 1 1 26 9 1 1 26 12 1
33 13 14 2 12 3 12 1 12 8 13 1 12 11 0 1 13 9 0 1 13 12 0
2 9 3 9 1 9 8 13 1 9 11 1 1 13 9 1 1 13 12 1
1 13 15 27 1 27 16 13
2 4 3 4 1 4 8 27 1 4 11 0 1 27 9 0 1 27 12 0
2 2 3 2 1 2 8 27 1 2 11 1 1 27 9 1 1 27 12 1
33 20 12 2 22 3 22 1 22 8 20 1 22 11 0 1 20 9 0 1 20 12 0
2 0 3 0 1 0 8 20 1 0 11 1 1 20 9 1 1 20 12 1
1 20 15 28 1 28 16 20
2 15 3 15 1 15 8 28 1 15 11 0 1 28 9 0 1 28 12 0
2 10 3 10 1 10 8 28 1 10 11 1 1 28 9 1 1 28 12 1
1 14 15 15 1 15 16 14 1 15 15 16 1 16 16 14
1 16 15 17 1 17 16 14 1 17 15 18 1 18 16 14
33 14 10 2 5 3 5 1 5 8 14 1 5 11 0 1 14 9 0 1 14 12 0
2 7 3 7 1 7 8 14 1 7 11 1 1 14 9 1 1 14 12 1
2 22 3 22 1 22 8 15 1 22 11 0 1 15 9 0 1 15 12 0
2 10 3 10 1 10 8 15 1 10 11 1 1 15 9 1 1 15 12 1
2 13 3 13 1 13 8 16 1 13 11 0 1 16 9 0 1 16 12 0
2 8 3 8 1 8 8 16 1 8 11 1 1 16 9 1 1 16 12 1
```

Figure 3.10 The DESCRIPTOR file

2. **The SYMT table**

As was shown in Chapter 2 (Table 1) the original SYMT table is built in the order that the variables are declared in the VOHL syntax of the circuit, while the

```
4 7 5 2 1 5 3 1 4 22 5 2 1 5 3 1 5 4 1 5 5 1
4 3 5 2 1 5 3 1 5 4 1 5 5 1 4 4 5 2 1 5 3 1 5 4 1 5 5 1
4 12 5 2 1 5 3 1 5 4 1 5 5 1 4 6 5 2 1 5 3 1 5 4 1 5 5 1
4 21 5 2 1 5 3 1 5 4 1 5 5 1 6 21 5 2 1 5 3 1
4 9 5 2 2 5 3 2 14 0 16 0 2 17 0 2 18 0 2 19 0 2
15 0 1 16 1 2 17 1 2 18 1 2 19 1 2 6 9 5 2 2 5 3 2
14 2 16 2 2 17 2 2 15 2 3 16 3 2 17 3 2
4 5 5 2 1 5 3 1 5 4 1 5 5 1 6 5 5 2 1 5 3 1
4 8 5 2 1 5 3 1 5 4 1 5 5 1 6 8 5 2 1 5 3 1
4 19 5 2 1 5 3 1 5 4 1 5 5 1 6 19 5 2 1 5 3 1 5 4 1 5 5 1
4 13 5 2 1 5 3 1 5 4 1 5 5 1 6 13 5 2 1 5 3 1 5 4 1 5 5 1
4 20 5 2 1 5 3 1 5 4 1 5 5 1 6 20 5 2 1 5 3 1 5 4 1 5 5 1
4 14 5 2 2 5 3 2 14 4 16 4 2 17 4 2
15 4 5 16 5 2 17 5 2 18 5 2 19 5 2 15 5 6 16 6 2 17 6 2
15 6 7 16 7 2 17 7 2 6 14 5 2 2 5 3 2 5 4 2 5 5 2
14 8 16 8 2 17 8 2 18 8 2 19 8 2 15 8 9 16 9 2 17 9 2 18 9 2 19 9 2
15 9 10 16 10 2 17 10 2 18 10 2 19 10
2 15 10 11 16 11 2 17 11 2 18 11 2 19 11 2
7  5 2 2 5 3 2 5 4 2 5 5 2 14 12 16 12 2 17 12 2 18 12 2 19 12 2
15 12 13 16 13 2 17 13 2 15 13 14 16 14 2 17 14 2
15 14 15 16 15 2 17 15 2 18 15 2 19 15 2
```

Figure 3.11 The DEFAULT DELAY file

```
4 3 5 2 2 4 3 5 3 4 4 3 5 4 2 4 3 5 5 3 4 12 5 3 3 4 4 5 4 3
4 4 5 3 2 6 5 5 2 2 4 5 5 5 3 4 6 5 2 3 4 6 5 3 2 4 6 5 4 4
```

Figure 3.12 The MODIFIED DELAY file

descriptor part and the default delay part of the SIMDATA file are built in the order that the circuit is described.

If the way to edit the circuit is by the modification of the SIMDATA, we need to know what position in the file is being described by the variable that we want to edit. In the original compiler we do not have this possibility because, after finishing the compilation, the system loses track of the position of the variables.

To find the correct position to be modified in the different files the system needs to know all the information about the space in each file that is occupied by each variable. To allow the presentation of this information the SYMT table was modified and presents the structure that can be seen in Table 3.

```
20 21 23 24 25 26 3 27 1 20 22 23 24 25 26 5 27 0
20 22 23 24 25 26 12 27 1
```

Figure 3.13 The INITIALIZATION file

```
28 0 0 A 29 0 0 28 1 0 A 28 1 1 1 29 1 1 28 2 0 B 29 2 2
28 3 0 A 28 3 1 8 28 3 2 5 29 3 3 28 4 0 A 28 4 1 5
29 4 7 30 31 5 32 50
```

Figure 3.14 The PRINTOUT file

The first four columns in the table are the same as the original table, except by the variables presented in different order. The order of the presentation of the variables was changed to allow them to be presented in the same order that they appear in the DESCRIPTOR and DEFAULT DELAY files.

The building of the table starts when the compiler reads the declarations of the VOHL syntax of the circuit. The variables are placed in the table and receive a descriptor number just as in the original compiler. When the compiler starts to read the description of the circuit it begins to swap the position of the variables so that their new positions in the table are the same as they appear in the files. For example, with this change we know that if we want to find the position of the variable A4 it will be the ninth descriptor in the descriptor file and the sixth descriptor in the default delay file because inputs do not appear in the default delay file.

The column despos (descriptor positions) presents the number of spaces that are occupied by each variable (descriptor) in the descriptor file. By this column, that is completed when the compiler writes the code for that descriptor in the file, we can see

TABLE 3

THE (MODIFIED) SYMT TABLE

name	descno	funcno	fanld	despos	delpos	ini num	pri num	pri val
A	0	0	4	11	0	0	1	1
A1	1	0	2	11	0	0	2	2
B	2	0	4	11	0	0	3	1
A5	7	5	2	23	8	0	5	2
A100	22	2	5	43	14	0	0	0
A85	3	1	0	43	14	1	4	3
A2	4	6	3	43	14	0	0	0
A10	12	4	2	43	14	3	0	0
A4	6	3	3	43	14	0	0	0
A19	21	13	1	71	22	0	0	0
A7	9	9	1	79	62	0	0	0
A8	10	9	3	0	0	0	0	0
A9	11	9	0	0	0	0	0	0
A3	5	8	2	71	22	2	0	0
A6	8	7	3	71	22	0	0	0
A17	19	11	1	91	28	0	0	0
A11	13	14	1	91	28	0	0	0
A18	20	12	1	91	28	0	0	0
A12	14	10	1	155	182	0	0	0
A13	15	10	1	0	0	0	0	0
A14	16	10	1	0	0	0	0	0
A15	17	10	0	0	0	0	0	0
A16	18	10	0	0	0	0	0	0

that, for example, the variable A3, that is an output of a three input NAND gate, has a code in the descriptor file that needs 71 spaces to be written. Also, as will be explained later, when the Editor try to find the descriptor in the file it needs only to sum the values in the column corresponding to the variables that appears in the table before the desired variable. For the case of the same variable we can see that the descriptor file has 401 spaces (11 + 11 + 11 + 23 + 43 + 43 + 43 + 43 + 43 + 71 + 79) occupied before starts the description of the variable.

An important point in the fulfillment of this column is the case of the multioutput gates. If a gate has n outputs it has n variables that will appear in the SYMT table. However, when the compiler makes the code corresponding to this gate all the connections between the descriptors will be written at the same time and, consequently, the system will only put a value in this column for the variable that corresponds to the first described output for this gate. A value of 0 will be put in all the other variables that correspond to outputs for the same gate. This can be seen in the Table 3 for the variables A7, A8 and A9 that are outputs of a SR gate. The only variable that has a value in the table is A7, with the number 79 in the column. The other two variables have the number 0 in this column. This says that this gate needs 79 spaces to completely describe its connections in the descriptor file.

The next column, delpos (delay positions), presents how many spaces are occupied by each descriptor in the DEFAULT DELAY file. All the observations that were made in the previous paragraphs for the despos column with respect to the descriptor file can be applied for this column with respect to the default delay file. Using this column we see that the variable A3 needs 22 spaces to describe its default delays and that it has 162 spaces (8 + 14 + 14 + 14 + 14 + 14 + 22 + 62) occupied in the default delay file before this variable starts to be described. Also for the multioutput gates the same concepts that were used in the previous column is used in this one.

The other 3 columns will also be used to find the position of the variables in the other files. The ini_num (initialization position) column shows the position where the variable appears in the initialization file, if it was an initialized variable. If the number in this column is zero the variable was not initialized and, consequently, does not appear in the file. The other numbers that appear in this column show the order that the variable appears. In the example shown, the order of the variables in the initialization part is: A85, A3 and A10.

The column pri_num (printout position) presents the position where the variable appears in the printout file, if a printout of the variable was requested in the VOHL description of the circuit. As the size of the printout code is function of the number of characters in the variable name, the column pri_val (printout spaces) presents the number of times that the code of printout appears in the printout file to represent that variable. If the number 0 appears in the "printout position" column, the variable is not an output of the system, i.e., no printout of this variable was requested. The other numbers present the position of the variable in the PTT file, with the numbers of the next column showing, in this case, how many times the code was repeated. In the example, the order of the printout code will be: A (with only 1 appearance of the code), A1 (with 2 appearances), B (1 appearance), A85 (3 appearances) and A5 (2 appearances).

3. **The DEL table**

The modified delay part of the SIMDATA file is built when the compiler reads the DEFINE part of the VOHL syntax. The delays will appear in the file in the same order that they were defined. For this reason, the EDITOR program needs to know in what order the delays will appear in SIMDATA to make the modifications.

The DEL table, that can be seen in Table 4, was created to allow the system to have a record of how the delays were placed in the file.

The meaning of the columns of this table are:

1. index - an index for the table
2. desc. - the descriptor number of the variable that will have its delay modified.
3. type - presents what delay is being modified, if it is the rise delay (code 10) or the rise delay (code 11).
4. input - input number which is having the delay modified.
5. output - output number which is having its delay modified
6. spaces - how many numbers will be printed in the file for this modification.

4. **The INP table**

The INP table, that can be seen in Table 5, was created to allow the system to store the inputs to each gate of the circuit. The actual system supports gates with up to 10 inputs, and, consequently, the table has space for 10 inputs. This table was also created to allow updates to the fanload of each gate when a modification (replace, insert, delete,. . .) is done in the circuit.

The meaning of each column in this table is:

1. name - name of the variable that is being described.

TABLE 4
THE DEL TABLE

index	desc.	type	input	output	spaces
0	3	10	0	0	5
1	3	11	0	0	5
2	3	10	1	0	5
3	3	11	1	0	5
4	12	11	0	0	5
5	4	10	1	0	5
6	4	11	0	0	5
7	5	10	2	0	5
8	5	11	1	0	5
9	6	10	0	0	5
10	6	11	0	0	5
11	6	10	1	0	5

2. nb. inp. - number of inputs of the gate
3. inp 1 - variable that goes in the input 1
4. inp 2 - variable that goes on input 2 (if input exists).
5. inp 3 - variable that goes in input 3 (if input exists).
6. inp 4 - variable that goes in input 4 (if input exists).
7. inp 5 - variable that goes in input 5 (if input exists)
8. inp 6 - variable that goes in input 6 (if input exists).
9. inp 7 - variable that goes in input 7 (if input exists).
10. inp 8 - variable that goes in input 8 (if input exists).
11. inp 9 - variable that goes in input 9 (if input exists).
12. inp 10 - variable that goes in input 10 (if input exists).

One important point to notice in this table is that for the multi-output gates case the only variable that is described is the last output presented in the description of the gate.

TABLE 5
THE INP TABLE

name	nb inp	inp 1	inp 2	inp 3	inp 4	inp 5	inp 6	inp 7	inp 8	inp 9	inp 10
A5	1	A4	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx
A100	2	A1	B	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx
A85	2	A	A19	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx
A2	2	A	A100	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx
A10	2	A5	A8	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx
A4	2	B	A3	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx
A19	3	A17	A18	A14	xxx	xxx	xxx	xxx	xxx	xxx	xxx
A9	3	A	A6	A4	xxx	xxx	xxx	xxx	xxx	xxx	xxx
A3	3	A100	A1	A2	xxx	xxx	xxx	xxx	xxx	xxx	xxx
A6	3	A4	A2	A100	xxx	xxx	xxx	xxx	xxx	xxx	xxx
A17	4	A12	A10	A6	B	xxx	xxx	xxx	xxx	xxx	xxx
A11	4	A10	A7	A2	B	xxx	xxx	xxx	xxx	xxx	xxx
A18	4	A100	A	A13	A8	xxx	xxx	xxx	xxx	xxx	xxx
A1	6	A3	A5	A100	A8	A11	A6	xxx	xxx	xxx	xxx

5. **The timing-wheel simulator**

The original timing-wheel simulator was designed to receive the entire SIMDATA file, read it, create the descriptors and make their interconnections. As the compiler program was modified to divide the SIMDATA into 5 different files, the simulator program also needed to be modified to rebuild the SIMDATA file before the simulator starts to read the file. The new timing-wheel simulator program can be seen in Appendix C.

IV. DESIGN, IMPLEMENTATION AND EXECUTION

A. THE DESIGN

As stated in Chapter 1 of this thesis, the Editor program allows fourteen operations to be done to any compiled circuit. These operations are:

1. Replace a gate - REPLACE;
2. Insert a gate - INSERT;
3. Delete a gate - DELETE;
4. Modify delay - ALTDEL;
5. Add a printout - ADDPRI;
6. Delete a printout - DELPRI;
7. Change an initialization value - ALTINI;
8. Insert an output - INSOUT;
9. Delete an output - DELOUT;
10. Change the delays in a type of gate - ALTGATE;
11. Insert input - INSINP;
12. Insert input and gate - INSINPG;
13. Delete an input - DELINP;
14. Delete an input and a gate - DELINPG;

The first step in the design process is to define what modifications are needed, how these modifications will be performed and the limitations imposed by each modification. Also in this first part it is necessary to consider the limitations imposed on the system by the editor.

The next subsections show the design decisions that drove the implementation of the EDITOR program.

1. Replace gate

This command will allow the substitution of one or more gates by another gate, with the restriction that the new gate must have the same number of outputs as the replaced gate(s). The system was designed with this feature because we supposed that when the user wants to replace a gate he/she does not want to modify the number of internal variables of the circuit and, because of that, the number of the outputs of the new and the old gates have to be the same, because the outputs of the gates are really the internal variables of the system.

In this part when a gate replaces another gate in the system, the replacement process maintains the default characteristics, i.e., the gate will have unmodified delays or initialization values, unless the user makes changes by using another editor command. Because of this the system needs to search all the files that compose the description of the system, eliminating all the references to the descriptor number of the old gate in the descriptor file, default delay file, modified delay file, and initialization file, and inserting the description of the new gate only in the descriptor file and in the default delay value. Another important consideration in this case is that as the variable name was not changed with the replacement of the gate and its descriptor number was not changed either.

2. **Insert and delete gates**

In this case the use of the insert or delete commands will change the number of internal variables of the circuit, by deleting or adding variables to the circuit. Also in this case, because the system has a different implementation, some of the remaining gates need to be changed. The important point about this change is that the only thing that will change in the gate is one or more of its inputs for the gates that are affected by the change of variables. When the system replaces this gate the only file that will be modified is the descriptor file, to allow the modification of the inputs, because all the other characteristics of the gate in the circuit remain unchanged.

3. **Add or delete inputs**

The system has two possibilities for the add and delete commands: one for the case where only the input itself is added to (deleted from) the circuit, and other for the case where together with the input another gate needs to be added to (deleted from) the circuit.

In the first case the system needs to add (delete) an input descriptor to (from) the descriptor file and after that make a modification in the circuit in the same way that was done in the insert/delete case (only the inputs of the gate changes). In the second case, if we are inserting an input and a gate, the system needs to add the input descriptor, a new gate in the descriptor file and in the default delay value, and to modify the inputs of the gates affected by the changes. If we are deleting an input and a gate the system needs to delete the input descriptor, delete the gate descriptor in all the files where it appears and modify the inputs of the gates affected by the deletion.

4. **Add or delete outputs**

The only modification that will be done in the circuit is the insertion (deletion) of the descriptor that describes the gate that is being inserted (deleted), because this insertion (deletion) will have no effect on the other gates of the circuit.

5. **Initial values**

The original system allows 3 values for the initialization of variables: 0, 1 or 2 (undefined). The EDITOR program will allow a fourth value for initialization that will be the number 3. When the program finds a request to initialize a variable with a 3 the program understands that this variable was already initialized and that no further action is wanted. In this case the part of the initialization file that describes the initial value of this variable will be deleted.

6. **Modify delays or gate**

Both commands have basically the same action, with the difference that the modify gate (ALTGATE) is more general than the modify delay (ALTDEL), because the modify delay command will change a specific delay of a specific gate, while the modify gate command will change a specific delay for all the gates of the referenced gate type that appears in the circuit. Any positive integer number could be used as a delay value in this case, but if the delay value was defined as 0 it will be understood by the program that this specific delay was already modified and the user wants it to return to its default value.

7. **The continue command**

This command was designed to allow several different modifications to be done in only one run of the EDITOR program. When the program finds this command (represented by the symbol #) it understands that the modification that it is doing is finished and that it will find another keyword to begin another modification. For example, with this command we can make a REPLACE, followed by an ALTDEL, followed by an ALTINI, etc.

8. **Syntax of the commands**

One of the important points in the design of the system was the definition of the syntax of each command. This is an important point because the EDITOR program needs to know the syntax for each command before it is written. The next section will present the syntax for all the commands for the EDITOR program.

9. **Limitations for the system**

Due to the way that the CADD program was designed, the EDITOR program is limited because it will not allow changes in user defined circuits as a block. When the system is compiled the first time, the user libraries (the sub-modules of the original circuit) are expanded to a lower level, until the expansion reaches the point where all of the description of the circuit is done by using system defined primitives. As we can see, in this case the system does not keep track of the user defined libraries. If the user decides to make changes in the circuit those changes must be based on the system defined primitives.

B. THE SYNTAX OF THE COMMANDS

The syntax for all the commands that are allowed in the EDITOR program is defined in this section. The description of each command will be done using the BNF notation.

1. **The REPLACE command**

```
REPLACE ;
    {
        <CKT>
    }
END;
```

where:

```
<CKT> ::= <var1> = <prim> ( <var2> ); <CKT> |
          <var1> = <prim> ( <var2> );
<var1> ::= output of the gate that is being replaced
<prim> ::= a primitive existing in the library of the system
<var2> ::= <var> , <var2> |
           <var> (the number of repetitions will be a
                  function of the number of inputs
                  to the gate)
<var> ::= one of the variables defined in the circuit.
```

2. **The INSERT command**

```
INSERT : <var3> ;
    {
        <CKT1>
        <CKT>
```

```
        }
    END;
```

where:

```
<var3> ::= <var4> , <var3> |
           <var4>
<var4> ::= name of the variable to be inserted
<CKT1> ::= <var4> = <prim> ( <var2> ) ; <CKT1> |
           <var4> = <prim> ( <var2> ) ;
<prim> ::= a primitive existing in the library of the system
<var2> ::= <var> , <var2> |
           <var> (the number of repetitions will be a
                  function of the number of inputs
                  to the gate)
<var> ::= one of the variables defined in the circuit.
<CKT> ::= the same definition as the REPLACE case.
```

3. **The DELETE command**

```
DELETE : <var5> ;
    {
       <CKT>
    }
END;
```

where:

```
<var5> ::= <var6> , <var5> |
           <var6>
<var6> ::= name of the variable to be deleted
<CKT> ::= the same definition as the REPLACE case.
```

4. **The ALTDEL command**

```
ALTDEL ;
    {
       <CKT2>
    }
END;
```

where:

```
<CKT2> ::= <var> : <OPT1> (<INP>,<OUT>) =
                   <VAL> ; <CKT2> |
```

<var> : <OPT1> (<INP>,<OUT>) =
<VAL>

<var> ::= one of the variables defined in the circuit.

<OPT1> ::= <RISEDEL> | <FALLDEL>

<INP> ::= gate input number corresponding to the path to be modified

<OUT> ::= gate output number corresponding to the path to be modified

<VAL> ::= value of the new delay of the path

5. **The ADDPRI command**

ADDPRI : <var7> ;
}
END;

where:

<var7> ::= <var8> , <var7> |
<var8>

<var8> ::= variable name of the variable to be printed

6. **The DELPRI command**

DELPRI : <var7> ;
}
END;

where:

<var7> ::= <var8> , <var7> |
<var8>

<var8> ::= variable name of the variable to be deleted from the printout

7. **The ALTINI command**

ALTINI ;
{
<CKT3>
}
END;

where:

<CKT3> ::= <var9> = <value> ; <CKT3> |
<var9> = <value> ;

< var9 > ::= name of the variable to have the initial value
changed

< value > ::= < 0|1|2|3 >

8. **The INSOUT command**

INSOUT : < var3 > ;
{
< CKT1 >
}
END;

where:

< var3 > ::= < var4 > , < var3 > |
< var4 >

< var4 > ::= name of the output to be inserted

< CKT1 > ::= < var4 > = < prim > (< var2 >) ; < CKT1 > |
< var4 > = < prim > (< var2 >) ;

< prim > ::= a primitiv existent in the library of the system

< var2 > ::= < var > , < var2 > |
< var > (the number of repetitions will be a
function of the number of inputs
to the gate)

< var > ::= one of the variables defined in the circuit.

9. **The DELOUT command**

DELOUT : < var5 > ;
}
END;

where:

< var5 > ::= < var6 > , < var5 > |
< var6 >

< var6 > ::= name of the variable to be deleted

10. **The ALTGATE command**

ALTGATE ;
{
< CKT5 >
}
END;

where:

<CKT5> ::= <prim> : <OPT1> (<INP>,<OUT>) =
<VAL> ; <CKT5> |
<prim> : <OPT1> (<INP>,<OUT>) =
<VAL>

<prim> ::= a primitive existent in the library of the system

<OPT1> ::= <RISEDEL> | <FALLDEL>

<INP> ::= gate input number corresponding to the path to be modified

<OUT> ::= gate output number corresponding to the path to be modified

<VAL> ::= value of the new delay of the path

11. **The INSINP command**

```
INSINP : <var10> ;
    {
       <CKT>
    }
END;
```

where:

<var10> ::= <var11> , <var10> |
<var11>

<var11> ::= name of the input to be inserted

<CKT> ::= the same definition as the REPLACE case.

12. **The INSINPG command**

```
INSINPG : <var10> ; <var3> ;
    {
       <CKT1>
       <CKT>
    }
END;
```

where:

<var10> ::= <var11> , <var10> |
<var11>

<var11> ::= name of the input to be inserted

<var3> ::= <var4> , <var3> |

<var4>

<var4> ::= name of the variable to be inserted

<CKT1> ::= <var4> = <prim> (<var2>) ; <CKT1> |
<var4> = <prim> (<var2>) ;

<prim> ::= a primitive existent in the library of the system

<var2> ::= <var> , <var2> |
<var> (the number of repetitions will be a
function of the number of inputs
to the gate)

<var> ::= one of the variables defined in the circuit.

<CKT> ::= the same definition as the REPLACE case.

13. **The DELINP command**

DELINP : <var12> ;
{
<CKT>
}
END;

where:

<var12> ::= <var13> , <var12> |
<var13>

<var13> ::= name of the input to be deleted

<CKT> ::= the same definition as the REPLACE case.

14. **The DELINPG command**

DELINPG : <var12> ; <var5> ;
{
<CKT>
}
END;

where:

<var12> ::= <var13> , <var12> |
<var13>

<var13> ::= name of the input to be deleted

<var5> ::= <var6> , <var5> |
<var6>

<var6> ::= name of the variable to be deleted

< CKT > ::= the same definition of the REPLACE case.

C. THE IMPLEMENTATION

In this section we describe how the editor was implemented to perform the different modifications in the circuit. The implementation is presented in a general form for each of the commands, but the program that was written to perform the edition will not be discussed here.

The programs that are necessary to perform the edition of any circuit are presented in the Appendices D and E and are sufficiently commented to allow an understanding of their behavior. The version that is presented in the appendices is the version designed for the IBM-PC/AT. With a few modifications, this version can also be used in the VAX system.

Appendix F presents the files and programs that are necessary to support the compilation and edition of any circuit. Appendix G presents the meaning of each one of the files presented in the other appendices, and also presents the reason why those files are necessary.

All the files and programs presented in the appendices were designed for the use of a IBM-PC AT or a compatible computer, with a hard-disk for storage of the files and programs and the capability of the using a RAM disk (virtual memory) to accelerate the program.

The next sub-sections present the implementation for each of the commands in the EDITOR program.

1. The replace command

After reading the replace command given by the user, the EDITOR looks in the SYMT table (Table 3) to verify the position of the variable being replaced. While searching the table, the system can determine the location of the first field of the descriptor that corresponds to the gate being replaced, by adding the values of the DESPOS column for the variables that appears before the variable that is the output for the gate being replaced. After finding the correct place, the system, knowing the number of spaces occupied by the gate being replaced, can skip the number of places that describe this gate. In the place that was occupied by the old gate, the system can now introduce the description of the new gate, and after this insertion it can copy the rest of the descriptor file.

The same procedure is used to replace the description of the default delay of the gate, with the difference that the system will work with the default delay file and will look for the DELPOS column, instead of the DESPOS column of the SYMT table.

As discussed in the beginning of this Chapter, the replacement of a gate implies that the new gate needs to have delays with their default values and an undefined initial value. To verify if modifications need to be done in the files that describe these parts, the system looks to the DEL table, to verify if the descriptor number of the gate being replaced appears there. If the descriptor does not appear the modified delay file is not modified. However, if the gate appears in this table this means that some delay modification from default exists in the old gate, and this value needs to be deleted. To delete this value the system adds the values that appear in the "spaces" column of the DEL table, until it reaches the gate being replaced. With this value it can find, in the modified delay file, in what position this gate is being described and delete it from the file. This operation is repeated until all the appearances of the gate in the table are deleted.

To verify if the gate was initialized in the previous simulation the system looks once more to the SYMT table (Table 3) and verifies, for the gate being replaced, the number that appears in the column init_position. If the number is 0 the variable was not initialized and the initialization file does not need to be modified. However, if the value is not 0 the variable is already initialized and needs to be deleted from the file. As each initialization description occupies 9 spaces in the file (as presented in the Chapter 2), the system knows that if the variable to be deleted has a number 3 in the column, two variables are described before it, and consequently the file has 18 numbers before the description of the variable. With this number, the system can find the beginning of the descriptor in the file, delete the next 9 numbers and copy the rest of the file. The only problem in this case is when the variable to be deleted is the first to be described in the file, because the description of the first variable is different from the others. In this case the system needs to delete the first 9 numbers from the file and change the second number of the next descriptor from 22 to 21, because this descriptor will now be the first one in the file.

During this process the tables (SYMT, DEL and INI) are corrected to represent the new state of the circuit after the replacement of the gate.

2. **The insert command**

The best way to put a new descriptor in the circuit is to place this descriptor at the end of the corresponding files. To allow this procedure the system, after receiving the command to insert a gate, looks to the SYMT table and, by using the values of the despos and delpos columns in the SYMT table, computes how many

numbers are placed in the descriptor file and in the default delay file. With those values the system can find the place in the files where the new descriptor will be placed and insert it in the correct place. The other files that describe the circuit are not modified because during the design it was assumed that when a gate is inserted in the circuit it is inserted with its default parameters and that no initial value is assigned to this gate. During the execution of this part the SYMT table, the DESC table and the INI table are modified to represent the new configuration of the system.

With the insertion of a new gate in the circuit some modifications need to be made in the circuit to allow the new gate to appear as input to some of the other gates. As was presented in the beginning of this Chapter, it was decided that during the design of the system, this modification would consist only of modifications in the inputs of some gates. Because of that, the only file that will be modified in this part is the descriptor file, and this modification will be done in the same way that the modification of this file was done in the replace case. In this part, the INI table is also modified to represent the new form of the circuit.

3. **The delete command**

In this part, it may be necessary that all the files that describe the circuit must be modified. The first file to be modified is the descriptor file, and to do this modification the system needs to know, as in the previous cases, where the descriptor that describes the variable to be deleted appears in the file. To find this position the system makes a search in the SYMT table to find the position of the variable, adding the values that appears in the DESPOS column to the variables that appear before the searched variable. With this value, the system can find the position of the variable in the file, and with the value of the DESPOS column for the gate being deleted the program knows how many numbers need to be deleted from the file. After deleting the corresponding number of places in the file the system copies the rest of the file.

The next step will be the deletion of the default delays, that correspond to this gate, from the default delay file. The procedure for this deletion is the same procedure that was used in the case of the descriptor file, with the difference that, in this case, the system computes the values in the DELPOS column and not in the DESPOS column.

The necessity of modifications in the other files is a function of the description of the circuit. The deletion of the descriptor from the modified delay file and from the initialization file, if necessary, is done in the same way that the deletion of those parts of the files in the replace case was done, and will not be repeated here. To verify if it is

necessary to modify the printout file, the system looks for the column print_position of the SYMT table, to determine if the variable is one of the variables that is being printed in the circuit. If the number that appears in this column is 0, the variable is not being printed and the file does not need to be modified. If the number is different from 0 the variable is being printed and the file needs to be changed. If the number in this column is 1 the variable is the first to be printed and to delete it from the file the system only verifies, in the next column of the table, the number of spaces that are occupied by the variable description. This verification is necessary because the number of spaces occupied by one variable in the printout file is function of the number of characters that composes its name. By verifying the column spaces, the system knows how many characters compose the variable name, and can make the transformation of this number to the number of places occupied by this variable in the file. This transformation is based in the fact that each character needs 4 spaces in the file and that each variable needs, beyond that, 3 more spaces to itself. In this case, if the number in the spaces column is 1 the variable needs 7 spaces to be described, if the number is 2 the variable needs 11 spaces, and so on. With this value the system deletes the corresponding numbers from the file and copies the rest of the file.

If the number in the print_position column is different from 0 and 1 the system needs to verify, in the same column of the table, the variables that have a print position number lower than the number for the variable that we want to delete. For these variables the system computes the number of spaces occupied by each one (by verifying the column spaces) and with the total value it can find the position where the description of the variable begins. The variable can then be deleted from the file, by computing the number of spaces occupied by the variable, and copying the rest of the file.

Also in this case some modifications are needed in the gates that remain in the circuit to compensate for the absence of one of the variables. This modification is the same as in the insert case, and the procedure to do this modification is the same as presented for that case.

4. **The altdel command**

In this case the only file that will be modified is the modified delay file. The modification of the delays of the variable can have 3 different cases to be treated.

The first case appears when the value of the delay is 0. In this case, the user wants to return the specified delay of the specified variable to its default value. To do

this deletion the system looks in the DEL table in the position that the descriptor number that describes the variable appears. When this number is found, the system verifies whether the number that appears in the type column of the table corresponds to the type of delay modification that we want to do (10 for rise delay and 11 for fall delay). If this value is different, the system restarts the search until it finds a place where the two numbers are the same (descriptor number and type). When the system finds this position the system determines if the input and the output numbers are the same as those that appear in the input and output columns of the table. When the system finds the position where all four numbers are the same it knows how many numbers are in the file before the description of this delay begin. With this value it can find the position of the descriptor in the file, delete the descriptor, and copy the rest of the file.

The second case occurs when the user wants a specified path of a gate to have a specific value of delay and the system can find the path in the DEL table(by verifying the four numbers presented before). In this case, the user wants a modification in a delay that is already modified and the only thing that the system needs to do is find the place where this descriptor begins, modify its value in the file, and copy the rest of the file.

The third case occurs when the system can not find the specified path in the DEL table. In this case the user wants to insert a modification of delay into the system and the system does this insertion in the end of the modified delay file, also putting the data corresponding to this path into the DEL table.

5. **The addpri command**

In this command the only file that is modified is the printout file. The printout file can be divided into two parts, one that presents the code of the variables to be printed and another, in the end of the file, that is the termination of the file.

The user uses this command when he/she wants to insert a printout of one of the system variables. The best place to insert the command is in the end of the portions of the file that contains the code for the printout. To do this insertion the system searchs the SYMT table to verify which variables are to be printed and how many spaces are occupied by the code. With this value the system can find where it will insert the code for the new printout and copy the file.

Because a printout is inserted into the system the SYMT table needs to be modified so a reference for this new printout appears in future utilizations of the

circuit. To accomplish the modification the system verifies, in the table, the position of the variable for which a printout was inserted. After finding the correct position, the system changes the value of the column print_position from a value 0 (no printout) to the value corresponding to the actual position of the variable in the printout file. The value depends on how many variables were printed before. Also the column print_spaces of the table needs to be modified to reflect the number of characters that compose the name of the variable.

6. **The delpri command**

In this part the only file that is modified is the printout file. The procedure to delete one of the printouts of the circuit is the same as described in the subsection 'The delete case' for the deletion of a printout, and because of that it will not be presented here.

7. **The altini command**

In this case the only file that is modified is the initialization file. The implementation of this command depends on the content of the command, that can be: deletion of a initial value, modification of a initial value, or insertion of an initial value.

If the system receives a 3 as the value in the command, it is understood as a order to delete an initial value from the file. To do this type of modification the system follows the same procedure that was presented in the replace command case.

If the value received is 0 (logical 0), 1 (logical 1) or 2 (undefined) the system searches the SYMT table to find the position where the variable appears. In this row the system looks in the column initial_position to verify the number that appears there.

If the number in this column is 0, the variable was not previously initialized, and the system is inserting a new initial value of a variable into the file. This insertion is done at the end of the file because the system knows how many variables are already initialized and consequently, how many numbers are in the file. With this value the system can find the position were it will insert the new initialization description.

If the number in the initial_position column is not 0, the variable has an initial value in the file already. In this case the system will modify the file. If the number in the column is n, the system knows that it will modify the code that is placed in the position 9*(n-1) spaces after the beginning of the file. With this value the system can find the place where the initialization descriptor of the variable begins, modify it, and copy the rest of the file.

8. **The insout case**

In this case the system is inserting an output into the system, i.e., inserting a gate that is really one of the outputs of the circuit and, consequently, no further modification is necessary in the system. The procedure to insert the output in the system is the same that was used in the INSERT case, with the difference that in this case the part corresponding to the modification of another gate in the circuit is not used.

9. **The delout case**

In this case the system is deleting an output from the system, i.e., deleting a gate that is really one of the outputs of the circuit and, consequently, no further modification is necessary in the system. The procedure to delete the output from the system is the same that was used in the DELETE case, with the difference that in this case the part corresponding to the modification of another gates in the circuit is not used.

10. **The altgate case**

With this command the value of the delay of some path of a specified gate will be modified in all the appearances of this gate in the circuit (global modification).

To do this modification the first step is to search the SYMT table to verify which variables are output from this type of gate. With the descriptor number of the variable the system now can apply the same procedure that was used in the ALTDEL case. The procedure is repeated until all the SYMT table was searched.

11. **The insinp case**

In this case the user wants to insert an input into the system, modifying the existing system, but without inserting another gate in the circuit.

To do this modification the system inserts the new input into the beginning of the descriptor file, copying the rest of the file after it. After that, the command makes the modifications in the circuit following the same procedure that was done in the modification part of the INSERT case.

12. **The insinpg command**

In this case the user wants to insert an input into the system, modifying the system not only because of the insertion of this input but also because of an insertion of a new gate into it.

To do this modification the system inserts the new input into the front of the descriptor file and after that inserts the new gate and modifies the circuit by using the same procedure that was used in the INSERT case.

13. **The delinp command**

In this case the user wants to delete one of the inputs of the circuit, changing only the gates that have this variable as input, without deleting any gate from the circuit.

To do that, the system searchs the SYMT table to find the beginning place of the descriptor for the input to be deleted, computing the value of the DESCV variable for the inputs that appear before it in the table. With this value the system knows the beginning position of the descriptor of the input and can delete it, copying the rest of the file. After that the system can modify the circuit by using the same procedure that was used in modification part of the DELETE case.

14. **The delinpg command**

In this case the user wants to delete an input from the circuit, modifying the circuit not only because of the deletion of this input, but also because of the deletion of a gate from the circuit.

To do that the system deletes the input following the procedure presented in the DELINP case. After that the system can delete the gate and modify the circuit by using the same procedure that was used in the DELETE case.

D. EXECUTION

1. Using the IBM PC/AT

All the files and programs that are necessary to run the compiler, editor and simulator programs are installed in the IBM PC/AT in the Computer Design Laboratory. All those files are grouped in the directory SIMD of that computer.

The steps to run the Compiler or the Editor programs, after the DOS prompt in the computer, are the following:

1. Type CD\SIMD - to change the directory to the desired directory
2. Type PATH3 - this command will allow all the paths containing the programs that support the compiler/editor programs be set up.
3. Using the KEDIT editor, write the VOHL descriptions for the circuit that will be compiled/edited, giving to that file a <filename> that will be used in all calls. In the case where the description is to be used with the Editor program, the complete filename for the file will be the same <filename> as the circuit to be modified, followed by the extension ED. As an example, suppose that we want to create a file that will describe the circuit that is shown in the Figure 1.2. We can give the name TEST to the file, and this name will be the <filename> for all the applications of the circuit. If after the compilation of the circuit we want to make any modification in the circuit we need to create a file called

TEST.ED, and that file is the file that will be used by the Editor program to make the modifications in the circuit. An example of a possible TEST.ED file is presented in the Figure 4.1 .

4. Using the KEDIT editor write the input file that will be used as an input to the circuit to be compiled/edited. The name of this file will be <filename>.IN, with <filename> being the same filename of the circuit that is being compiled/edited.

5. If the compilation of the entire circuit, or the simulation of the circuit with its compilation is desired, type:

   MODEL2A <filename> <option>

   With this command the system will compile or simulate (depending of the option that was determined by the user) a circuit that is stored in a file with the same <filename>. The options for the command are:

   c - compile the circuit, without simulation.
   s - simulate a circuit already compiled
   e - compile and simulate a circuit

   For the last two cases an input file is necessary for the correct function of the system. In both cases the results will appear in the file <filename>.OUT. After running the program (in any of the 3 cases) the DESCT table, the SYMT table, the DEL table, the INI table, and all the files that describe the circuit (descriptor, default delay,etc...) will be stored in the hard-disk to allow reuse.

6. If it is desired to edit the circuit, or to simulate the circuit with editing, type:

   MODEL3B <filename> <option>

   With this command the system will edit or simulate a circuit that is stored in a file with the same <filename>. The circuit needs to be previously compiled. The options for this command are:

   c - edit the circuit, without simulation.
   s - simulate a circuit already compiled/edited
   e - edit and simulate a circuit

   For the last two cases an input file is necessary for the correct function of the system. In both cases the results will appear in the file <filename>.OUT. After running the program (in any of the 3 cases) the DESCT table, the SYMT table, the DEL table, the INI table, and all the files that describe the circuit (descriptor, default delay,etc...) will be stored in the hard-disk to allow a later use.

2. Using the VAX-UNIX

The necessary programs for the execution of the compiler, editor and simulator programs are also found in the VAX computer at the Naval Postgraduate School. Here there is no possibility for the use of an unique command to execute the file. Instead, each operation needs to be called in a different way. There are 3 different executable files each one being called in a different way (CADD2 for the compiler, SIMULA for the simulator and CEDT for the editor).

To compile the circuit, the command is:

```
REPLACE ;
{
        A5 = AND (A4, A1) ;
        A10 = ORTHREE (A5, A8, A3) ;
        A6 = AND (A4, A2) ;
#

ALTINI ;
{
        A10 = 3 ;
        A3 = 1 ;
        A19 = 0 ;
#

ADDPRI : A11, A8, A2 ;
#

ALTDEL ;
{
        A2 : RISEDEL(1,0) = 3 ;
        A6 : FALLDEL(0,0) = 2 ;
        A6 : RISEDEL(1,0) = 3 ;
        A17 : FALLDEL(3,0) = 3 ;
#

INSOUT : A20 ;
{
        A20 = ANDTHRE (A, A18, A11) ;
#

INSINPG : A21; A22 ;
{
        A22 = OR (A21, EN) ;
        A17 = ANDFOUR (A12, A10, A6, A22) ;
        A1 = OR (CLK, A21) ;
        A4 = NAND (A22, A3) ;
}
END;
```

Figure 4.1 The TEST.ED file

CADD2 <filename>

where filename is the file that contains the circuit to be compiled.

To edit the circuit, the command is:

CEDT <filename>

where filename is the file that contains the circuit to be edited.

To run the simulation of the circuit alone the command is:

SIMULA <filename>

where filename is the file that contains the circuit to be simulated. In this case a file <filename>.in, with the input values of the circuit, needs to be created to allow the simulation of the circuit, and a file <filename>.out will be created by the system with the outputs for the circuit.

V. PERFORMANCE

To verify the correctness of the EDITOR program and evaluate its performance several tests were run with different circuits, each run testing one of the capabilities of the program.

If the objective of the tests were only to verify the correctness of the program any set of tests that cover all the possibilities of the program could be considered a good test, but when the test must also evaluate the program some points need to be considered.

The time that the EDITOR program will take to run a specific test will depend on two characteristics: what is the modification that we want to introduce into the circuit and what is the size of the files (descriptor file, default delay file, modified delay file, initialization file and printout file) that describe the circuit. The descriptor file and the default delay file have a size that can be considered directly proportional to the size of the circuit. The size of the other 3 files will be a function of the special characteristics that the user gives to the circuit, and can be either empty or large, depending on the description of the circuit given by the user. The time that the compiler program needs to generate those files is also a function of the size of the circuit and its characteristics, with the time increasing when the size or the number of characteristics increases.

The evaluation of the EDITOR program can be done by measuring the time that is needed to perform some modification in the circuit using the program and by comparing this time with the time that would be necessary to compile the entire circuit with the modification inserted. At this point the first question appears. What is the better way to do this comparison? It is better to use a large circuit with a lot of characteristics inserted, a large circuit with few characteristics inserted, or a small circuit with few characteristics? It is almost impossible to answer this question, because the answer will be different from case to case, depending of what we want to modify in the circuit. For example, if we want to modify the initial values of the variables it is better to have a large circuit with few characteristics, particularly if it has few initial values in the original circuit, because the time that will be needed to rewrite the initialization file will be small with respect to the time to write all the files. But, if we

want to insert a gate into the circuit the time that will be used for the EDITOR program will be the same for each circuit and will be dependent only of the size of the circuit. However, the performance with respect to the compiler program will be different depending on whether the circuit has few (or none) delay modifications, initializations, and printouts or if it has many of those characteristics.

Another question concerns the necessity of recompiling the circuit with the modification inserted to allow a correct comparison with the EDITOR program. If we think in real terms, each of the tests that are done in the circuit really need to be two tests, one using the EDITOR program and the other using the recompilation of the circuit with the modification. However, the time required to perform all the necessary tests could be incompatible with the time available for the preparation of this thesis, and because of that, some of the recompilations were done, but in the cases where the modifications were similar the recompilations of the entire circuit was done only once, with this time being considered the default for the similar cases. This was done because when we do a compilation of the entire circuit the factors that determine the time needed for the compilation are the size of the circuit and the number of different characteristics in the circuit. To understand that suppose that we have a circuit with n gates. If we insert 3 gates and replace 1 gate in this circuit the circuit will have $n+3$ gates and the same characteristics. If, in the original circuit, we insert 3 gates and make 3 modifications the number of gates that the compiler program will found is $n+3$, and the characteristics of the circuit will be the same. As we can see the time for the edition of the circuit will be different in the two cases, because we are inserting and modifying a different number of gates, but the time for compilation of the entire circuit will be the same, because the number of gates and the characteristics of the circuit will be the same.

To perform the tests of the Editor program two circuits were used. The first circuit was the ALU circuit, that is presented in the Appendix H, and the second circuit was the circuit that is presented in Figure 1.2 . The ALU circuit is composed of 142 gates, but has no delay modification, few initializations, and printouts. The second circuit is composed of 14 gates, but has some delay modifications, few initializations, and a medium number of printouts.The first step in the test of the EDITOR program is to compile both circuits and verify the time that is needed by each one to complete the compilation. The ALU program needed 4 minutes and 35.38 seconds to complete the compilation, while the second circuit, that we will call TEST circuit, needed 40.94 seconds to perform the compilation.

The measurement of the performance of the editor was made by the computation of the ratio between the time needed by the editor to perform the corresponding modification and the time needed by the compiler, in percentage. The values found in the tests are presented in the last column of the tables that are presented in this chapter. The equation for the computation of this value is presented in the Equation 5.1

$$\text{ratio} = \frac{\text{edit time}}{\text{compilation time}} \times 100\% \qquad \text{(eqn 5.1)}$$

More than 350 tests were performed to evaluate the performance of the Editor program and the results will be presented in the following sections.

A. THE REPLACE CASE

When the replacement of gates is done in the circuit the time that will be needed to recompile the entire circuit is, in the worst case, the same that was necessary to compile the original circuit. The worst case is when none of the gates being replaced had delay modifications or initial values in the original circuit because in this case nothing will be deleted from the files. If some of the gates had any of the mentioned characteristics in the original program some files will be small, and consequently, the system will need less time to do the compilation. This difference in time is not very large and, for the effects of the test, the time of recompilation will be considered as being the same time necessary for the compilation of the original circuit.

The Table 7 presents the results of the tests done with the ALU circuit for the replace gate cases. The first column of the table presents the number of gates that are being replaced in the circuit, with the second and third columns presenting the amount of time needed for the edition and recompilation of the circuit, respectively. The last column of the table presents the comparison (in %) between the time for the edition and the time for recompilation of the circuit.

As we can see we can replace approximately 5% of the number of gates in the circuit using the editor program in an amount of time less than the time needed for the recompilation of the entire circuit. This result could be considered satisfactory, since the ALU circuit can be considered, perhaps, the worst case for the replacement of the

TABLE 7

THE REPLACE CASE
FOR THE ALU CIRCUIT

gates replaced	edit time min sec	compilation time min sec	ratio %
1	01 00.93	04 35.38	22.13
2	01 31.92	04 35.38	33.38
3	02 16.67	04 35.38	49.63
4	02 41.36	04 35.38	58.60
5	03 09.35	04 35.38	68.76
6	03 38.15	04 35.38	79.22
7	04 07.32	04 35.38	89.81

gates, having no modification of delays and very few initializations, factors that certainly would increase the time needed for the compilation of the circuit and that could allow a great number of gates to be replaced in the time that was needed by the compiler program for the compilation of the circuit.

Table 8 presents the results of the tests done with the TEST circuit for the replace gate cases. This table has the same format as the table presented for the ALU circuit.

As we can see this case is better than the ALU case, because in the time needed for the recompilation of the circuit we can replace almost 29% of the gates of the circuit. This could be considered an excellent result, because is not normal a replacement for this amount of gates in a circuit.

B. THE INSERT CASE

In this case two parameters are important in the measurement of the performance of the Editor program: how many gates are being inserted and how many gates will be replaced due to those insertions. The measurement of the performance of the program in this case was done by comparing the amount of time needed by the Editor program to modify the circuit and the time needed by the recompilation of the

TABLE 8

THE REPLACE CASE
FOR THE TEST CIRCUIT

gates replaced	edit time min sec	compilation time min sec	ratio %
1	00 22.95	00 42.64	53.82
2	00 28.30	00 42.64	66.37
3	00 34.10	00 42.64	79.97
4	00 39.32	00 42.64	92.21

circuit that is composed of the n original gates plus the number of gates that are being inserted.

The Table 9 presents the result of the tests for the ALU circuit. The table is basically the same as the replace case with one more column added to the table to represent the number of gates inserted in the circuit.

As we can see, the Editor program allows that 1 gate be inserted and 8 modified or 7 gates be inserted and 1 modified in the same time that the compiler program needs to compile the entire circuit. These numbers show that we can insert/replace more than 5% of the gates of the circuit, that is very powerful.

The Table 10 presents the result of the tests for the TEST circuit. The table format is the same of the ALU case.

In this case the Editor program allows the insertion/replacement of up to 6 gates in less time than the time necessary for the compilation of the entire circuit. Those numbers show that we can change almost 43% of the gates of the entire circuit without the necessity of recompilation. This is really an excellent result.

There is a great difference in the number of gates changed in both circuits. This difference can be explained by the size of the different files that describe the circuits. In the ALU circuit we do not have the modified delay file, the initialization file is very small, and the descriptor file and the default delay files are very large. As the insertion and replace cases works basically with these two files and as the other files are almost non-existent, the Editor program does not take as much time to copy the other files back and forth. If the circuit had delay modifications and more initializations certainly

TABLE 9

THE INSERT CASE
FOR THE ALU CIRCUIT

gates inserted	gates replaced	edit time min sec	compilation time min sec	ratio %
1	1	01 24.82	04 39.12	30.39
1	2	01 50.04	04 39.12	39.42
1	3	02 16.67	04 39.12	48.96
1	4	02 41.36	04 39.12	57.81
1	5	03 07.42	04 39.12	67.15
1	6	03 32.27	04 39.12	76.05
1	7	03 58.09	04 39.12	85.30
1	8	04 23.80	04 39.12	94.51
2	1	01 58.35	04 41.67	42.02
2	2	02 23.14	04 41.67	50.82
2	3	02 49.79	04 41.67	60.28
2	4	03 14.07	04 41.67	68.90
2	5	03 40.04	04 41.67	78.12
2	6	04 05.84	04 41.67	87.28
2	7	04 31.86	04 41.67	96.52
3	1	02 30.99	04 44.08	52.41
3	2	02 55.45	04 44.08	60.90
3	3	03 22.88	04 44.08	70.42
3	4	03 49.68	04 44.08	79.73
3	5	04 16.18	04 44.08	88.93
3	6	04 42.96	04 44.08	98.22
4	1	03 03.85	04 46.37	64.20

TABLE 9
THE INSERT CASE (CONT'D.)

gates inserted	gates replaced	edit time min sec	compilation time min sec	ratio %
4	2	03 31.09	04 46.37	73.71
4	3	03 58.50	04 46.37	83.28
4	4	04 25.72	04 46.37	92.79
5	1	03 38.01	04 48.72	75.51
5	2	04 06.16	04 48.72	85.26
5	3	04 34.47	04 48.72	95.06
6	1	04 13.21	04 50.85	87.06
6	2	04 42.21	04 50.85	97.03
7	1	04 50.25	04 53.01	99.06

the number of gates that could be replaced would be greater. This can be seen in the TEST circuit. In this circuit, even with the descriptor and default delay files being the largest files of the circuit, the other 3 files have a reasonable size and this will influence the number of gates that we can change, as we can see by the results obtained during the tests.

C. THE DELETE CASE

Also in this case, two parameters are important in the measurement of the performance of the Editor program: how many gates are being deleted and how many gates will be replaced due to those deletions. The measurement of the performance of the program in this case was done by comparing the amount of time needed by the Editor program to modify the circuit and the time needed to recompile the circuit that is composed of the n original gates less the number of gates that are being deleted.

Table 11 presents the result of the tests for the ALU circuit. The table is basically the same as the insert case with the column gates inserted replaced by the column gates deleted.

TABLE 10

THE INSERT CASE
FOR THE TEST CIRCUIT

gates inserted	gates replaced	edit time min sec	compilation time min sec	ratio %
1	1	00 22.88	00 43.49	52.61
1	2	00 28.69	00 43.49	65.97
1	3	00 33.18	00 43.49	76.29
1	4	00 37.21	00 43.49	85.56
1	5	00 41.36	00 43.49	95.10
2	1	00 30.97	00 45.39	68.23
2	2	00 34.21	00 45.39	75.37
2	3	00 38.98	00 45.39	85.88
2	4	00 43.23	00 45.39	95.24
3	1	00 36.18	00 46.44	77.91
3	2	00 40.60	00 46.44	87.42
3	3	00 45.31	00 46.44	97.57
4	1	00 41.66	00 48.33	86.20
4	2	00 46.49	00 48.33	96.19
5	1	00 47.38	00 50.41	93.99

As we can see the Editor program allows 1 gate to be deleted and 8 modified or 6 gates be deleted and 1 modified in the same time that the compiler program needs to compile the entire circuit. These numbers show that we can delete/replace almost 5% of the gates of the circuit.

Table 12 presents the result of the tests for the TEST circuit. The table format is the same as the ALU case.

As we can see, in this case the Editor program allows the deletion and replacement of until 4 or 5 gates in less time than the time necessary for the

TABLE 11

THE DELETE CASE
FOR THE ALU CIRCUIT

gates deleted	gates replaced	edit time min sec	compilation time min sec	ratio %
1	1	01 26.31	04 35.07	31.38
1	2	01 56.14	04 35.07	42.22
1	3	02 16.17	04 35.07	49.50
1	4	02 42.47	04 35.07	59.06
1	5	03 05.74	04 35.07	67.52
1	6	03 30.85	04 35.07	76.65
1	7	03 55.29	04 35.07	85.54
1	8	04 20.11	04 35.07	94.56
2	1	02 02.98	04 33.58	44.95
2	2	02 23.35	04 33.58	52.40
2	3	02 47.56	04 33.58	61.25
2	4	03 11.74	04 33.58	70.09
2	5	03 36.21	04 33.58	79.03
2	6	04 00.54	04 33.58	87.92
2	7	04 25.15	04 33.58	96.92
3	1	02 39.71	04 32.31	58.65
3	2	03 03.01	04 32.31	67.21
3	3	03 29.01	04 32.31	76.75
3	4	03 53.16	04 32.31	85.62
3	5	04 18.57	04 32.31	94.95
4	1	03 16.43	04 30.58	72.60
4	2	03 39.24	04 30.58	81.03
4	3	04 02.39	04 30.58	89.58

TABLE 11

THE DELETE CASE (CONT'D.)

gates deleted	gates replaced	edit time min sec	compilation time min sec	ratio %
4	4	04 25.24	04 30.58	98.03
5	1	03 42.84	04 28.97	82.85
5	2	04 05.29	04 28.97	91.20
6	1	04 19.65	04 27.02	97.24

TABLE 12

THE DELETE CASE
FOR THE TEST CIRCUIT

gates deleted	gates replaced	edit time min sec	compilation time min sec	ratio %
1	1	00 25.71	00 42.26	60.84
1	2	00 30.16	00 42.26	71.37
1	3	00 33.24	00 42.26	78.66
1	4	00 37.09	00 42.26	87.77
1	5	00 41.00	00 42.26	97.02
2	1	00 31.59	00 41.33	76.43
2	2	00 35.81	00 41.33	86.64
2	3	00 40.07	00 41.33	96.95
3	1	00 37.31	00 39.67	94.05

compilation of the entire circuit. These numbers show that we can change almost 30% of the gates of the entire circuit without the necessity for recompilation.

D. THE ALTDEL CASE

In the ALTDEL case we have three distinct possibilities; insertion of delays, modification of delays already modified, or return to default delays.

In this case, the tests were not done by increasing 1 variable at a time because, as the only file that is modified in this case is the modified delay file, the variation of time for increasing 1 variable is very small. The Table 13 presents the results of the tests executed for the ALU circuit. The table presents the type of modification made in the circuit, the number of paths whose delays were affected by the modification, the time necessary to do the modification with the Editor program, and the comparison with the time needed to do the compilation of the circuit with the modification inserted.

In the column modification type (MODIF. TYPE) a code was used to represent the modification that was done to the delays. This code is:

1. I - insertion of delays
2. M - modification of delays
3. D - return to default

Table 14 presents the results of the tests executed for the TEST circuit. The format of this table is the same as the table that presented the results for the ALU circuit.

As we can see the Editor program is a powerful tool when the user needs to modify the delays of the circuit that is being simulated. The system allows the insertion of delays in a number of paths that is almost equal to 1.5 times the number of gates that compose the circuit. For the case of deletion of delays the number of paths that we can work with actually outnumber the number of gates existing in the circuit, and only in the modification case does this number fall to one half of the number of gates. The difference in time in each of these cases is due to the size of the files that the system is working with. We can delete(insert) more delays than modify delays because in the first case the modified delay file starts big(small) and starts to decrease(increase) while the system is deleting(inserting) delays and, consequently, the average time to work with a specific number of delays is lower than the time to modify the same number of delays. On the other hand, for the same case the modified delay file will always have the same size.

E. THE ADDPRI CASE

As the MultiSim package presents the output as a table of results, the number of printouts that are allowed in the system are bounded by the size of the screen. As the

TABLE 13

THE ALTDEL CASE
FOR THE ALU CIRCUIT

modif. type	paths changed	edit time min sec	compilat. time min sec	ratio %
I	1	00 22.15	04 36.02	8.02
I	3	00 26.43	04 36.30	9.57
I	6	00 28.11	04 37.13	10.14
M	6	00 28.52	04 37.13	10.29
D	6	00 27.22	04 35.38	9.88
I	12	00 30.47	04 39.01	10.92
M	78	04 04.36	05 15.39	77.48
I	90	01 48.14	05 31.86	32.59
D	90	01 48.41	04 35.38	39.37
I	156	04 13.71	06 12.31	68.14
D	156	04 14.63	04 35.38	92.46

norm, we can have a maximum of 80 columns on the screen and the maximum number of printouts allowed is 18, if all the variables have a name with 2 characters, because the first column in the printout, which is the time, occupies 4 columns, and we need to have one column of separation between each name.

Because of this, the tests that were performed were limited to 17 additional printouts in the circuit.

Following the statements presented before, the EDITOR program was tested inserting 17 variables in the printout of the ALU circuit, and it needed 37.31 sec, that is only 13.51% of the 4 minutes and 36.15 seconds needed to compile the entire circuit with 18 printouts. When the program was tested on the TEST circuit it needed 27.24 seconds to add the 17 printouts in the circuit, what is 63.67% of the 42.83 seconds needed to compile the entire circuit.

TABLE 14

THE ALTDEL CASE
FOR THE TEST CIRCUIT

modif. type	paths changed	edit time min sec	compilat. time min sec	ratio %
I	1	00 17.35	00 42.64	40.69
I	11	00 19.87	00 44.73	44.42
M	11	00 31.06	00 44.73	69.44
I	16	00 23.98	00 45.21	53.04
M	20	00 33.14	00 45.86	72.26
D	20	00 25.87	00 42.64	60.67

With the limitations imposed by the actual output of the simulator, the case for added printouts in the circuit is always better with the Editor program.

F. THE DELPRI CASE

In this case the limitations that appeared in the add printout case also apply and, because of that, the system was tested for the worst case of 17 printouts deletions.

In the ALU circuit the Editor program needed 36.85 seconds to delete the 17 printouts, that is only 13.38% of the time needed by the system to compile the entire circuit.

In the TEST circuit the system needed 26.60 seconds to delete the 17 printouts, an amount of time that corresponds to 63.89% of the 41.63 seconds needed by the system to compile the entire circuit.

G. THE ALTINI CASE

In the ALTINI case we can have three distinct possibilities that are: initialization of variables that are not initialized yet; modification of initialization values of variables that are already initialized; deletion of variables from the initialization list,i.e., the variable was initialized before and the user wants that it be undefined.

In this case, the tests were not done by increasing 1 variable each time because, as the only file that is modified in this case is the initialization file, the variation of time to increase 1 variable is very small. Table 15 presents the results of the tests executed for the ALU circuit. The table presents the type of modification made in the circuit, the number of variables that were affected by the modification, the time necessary to do the modification with the Editor program, and the comparison with the time needed to do the compilation of the circuit with the modification inserted.

In the column modification type (MODIF. TYPE) a code was used to represent the type of modification to the variables. This code was:

1. I - insertion of initial values into new variables
2. M - modification of initial values of variables already initialized
3. D - deletion of initial values

The column total presents the number of initial value assignments in the initialization file at the end of the user command. This information is important because the size of the initialization file is the predominant factor in this command's performance. Due to the size of the file, the modification of the same number of initial values can have different times, depending on whether the initialization file is small or not.

Table 16 presents the results of the tests executed for the TEST circuit. The format of this table is the same as the table that presented the results for the ALU circuit.

As we can see the Editor program is a powerful tool when the user needs to modify the initial values of the variables in the circuit being simulated. The system allows the insertion/deletion of initial values in a number of variables that could be, in some of the cases, equal to the number of gates in the circuit. From the tables, for small circuits the Editor program is capable of inserting, modifying, or deleting the initial values for all the variables in the circuit in less time than the time needed for compilation of the circuit. For large circuits we can insert or delete a number of initial conditions that will be almost equal to the number of gates in the circuit, but we cannot modify a large number. In any event we can modify the initial values in almost one half of the gates in the circuit. The difference in time for each of these cases is due to the size of the files. The number of files where we can delete(insert) more initial values is more than the number of files we can modify, because in the first case the initialization file starts big(small) and starts to decrease(increase) while the system is

TABLE 15

THE ALTINI CASE
FOR THE ALU CIRCUIT

modif. type	var. changed	total	edit time min sec	compilat. time min sec	ratio %
I	1	2	00 27.05	04 36.02	9.80
M	20	142	02 14.57	05 12.21	43.10
D	30	112	02 52.30	04 59.83	57.47
I	64	65	01 53.78	04 43.08	40.19
M	64	65	03 08.73	04 43.08	66.67
D	64	1	01 53.66	04 35.38	41.27
I	111	112	04 39.66	04 59.83	93.27
D	110	1	04 34.81	04 35.38	99.79

TABLE 16

THE ALTINI CASE
FOR THE TEST CIRCUIT

modif. type	var. changed	total	edit time min sec	compilat. time min sec	ratio %
I	1	6	00 18.11	00 42.64	42.47
I	11	16	00 20.40	00 45.17	45.16
M	11	16	00 32.93	00 45.17	72.90
I	16	21	00 25.51	00 46.83	54.47
M	20	21	00 34.92	00 46.83	74.57
D	20	1	00 27.20	00 42.11	64.59

deleting(inserting) initial values. Consequently, the average time to work with a specific number of initial values is lower than the time to modify the same number of values.

H. THE INSOUT CASE

As the system worked with the insertion of gates the tests were done once more by inserting one gate at a time.

Table 17 presents the results of the tests for the ALU circuit. The first column of the table presents the number of gates being inserted in the circuit, with the second and third columns presenting the amount of time needed for the edition and recompilation of the circuit, respectively. The last column of the table presents the comparison (in %) between the time for the edition and the time for recompilation of the circuit.

TABLE 17
THE INSOUT CASE
FOR THE ALU CIRCUIT

gates inserted	edit time min sec	compilation time min sec	ratio %
1	00 57.63	04 39.12	20.65
2	01 30.80	04 41.67	32.24
3	02 02.82	04 44.08	42.63
4	02 36.80	04 46.37	54.75
5	03 09.25	04 48.72	65.55
6	03 42.07	04 50.85	76.35
7	04 15.53	04 53.01	87.21
8	04 49.34	04 55.18	98.02

As we can see the number of outputs that can be inserted in the circuit is almost equal to 6% of the number of gates that compose it. This result can be considered satisfactory because it is not normal for a user to insert a large number of outputs in the circuit in one change.

Table 18 presents the results of the tests done with the TEST circuit for the insert output case. This table has the same format as the table presented for the ALU circuit.

TABLE 18

THE INSOUT CASE
FOR THE TEST CIRCUIT

gates inserted	edit time min sec	compilation time min sec	ratio %
1	00 22.00	00 43.49	50.59
2	00 27.63	00 45.39	60.87
3	00 32.51	00 46.44	70.00
4	00 38.08	00 48.33	78.79
5	00 44.37	00 50.41	88.02

As we can see, this case is better than the ALU case, because for the same time needed for the recompilation of the circuit we can insert almost 36% of the gates of the circuit. Because it is not normal to insert this amount of outputs in a circuit, this can be considered an excellent result.

I. THE DELOUT CASE

In this case, as the system worked with the deletion of gates the tests were done once more by deleting one gate at a time.

Table 19 presents the results of the tests for the ALU circuit. The first column of the table presents the number of gates that are being deleted from the circuit, with the second and third columns presenting the amount of time needed for the edition and recompilation of the circuit, respectively. The last column of the table presents the comparison (in %) between the time for the edition and the time for recompilation of the circuit.

As we can see the number of outputs that can be deleted in the circuit is almost equal to 6% of the number of gates that compose it. This result can be considered satisfactory because is not normal for a user to delete a large number of outputs in the circuit in one change.

Table 20 presents the results of the tests done with the TEST circuit for the delete output case. This table has the same format as the table presented for the ALU circuit.

TABLE 19

THE DELOUT CASE
FOR THE ALU CIRCUIT

gates deleted	edit time min sec	compilation time min sec	ratio %
1	00 59.53	04 35.07	21.64
2	01 31.61	04 33.58	33.49
3	02 04.14	04 32.31	45.59
4	02 35.91	04 30.58	57.62
5	03 07.63	04 28.97	69.76
6	03 38.44	04 27.02	81.81
7	04 09.36	04 25.65	93.87

TABLE 20

THE DELOUT CASE
FOR THE TEST CIRCUIT

gates deleted	edit time min sec	compilation time min sec	ratio %
1	00 24.00	00 42.26	56.79
2	00 28.97	00 41.33	70.09
3	00 35.21	00 39.67	88.76

The TEST case is better than the ALU case, because in the time needed for the recompilation of the circuit we can delete almost 22% of the gates of the circuit.

J. THE ALTGATE CASE

This case has practically the same function as the ALTDEL case, with the only difference that this case works with the general gates of the circuit, instead of the

variables of the circuit. As a consequence, the results obtained in these tests are almost the same of the ALTDEL case, and, because of this the results will not be repeated.

K. THE INSINP CASE

In this case two parameters are important in the measurement of the performance of the Editor program: how many inputs are being inserted and how many gates will be replaced due to those insertions. The measurement of the performance of the program in this case was done by comparing the amount of time needed by the Editor program to modify the circuit and the time needed by the recompilation of the circuit.

Table 21 presents the result of the tests for the ALU circuit. The table is basically the same as the insert case with the gates inserted column replaced by the inputs inserted column.

The Editor program allows 1 input to be inserted and 8 gates to be replaced or 7 inputs to be inserted and 2 gates to be replaced in an amount of time lower than the time needed by the compiler program to recompile the entire circuit. These values could be considered very good, because we are modifying more than 5% of the number of gates in the system in a time lower than the compilation time.

Table 21 does not have the results of the tests for the insertion of 7 inputs and the modification of 1 gate. This test was not done because in the actual situation of the system, there are no primitive with 7 inputs and, consequently, it is impossible that 7 inputs be inserted in the system with only 1 gate being modified. When 7 inputs are inserted in the circuit at least 2 other gates need to be modified, and this is the reason that the test for 7 inputs and 1 gate was not studied.

Table 22 presents the result of the tests for the TEST circuit. The table format is the same of the ALU case.

In this case the Editor program allows that 1 input be inserted and 6 gates modified or 5 inputs be inserted and 1 gate modified in an amount of time lower than the time needed to recompile the entire circuit. This is a reasonable result because we are modifying almost 50% of the number of gates in the circuit in a time lower than the time to recompile the entire circuit.

L. THE INSINPG CASE

Three parameters are important in the measurement of the performance of the Editor program: the number of inputs being inserted, the number of gates being

TABLE 21

THE INSINP CASE
FOR THE ALU CIRCUIT

input inserted	gates replaced	edit time min sec	compilation time min sec	ratio %
1	1	01 16.46	04 35.80	27.72
1	2	01 43.31	04 35.80	37.46
1	3	02 08.32	04 35.80	46.53
1	4	02 34.49	04 35.80	56.02
1	5	03 00.83	04 35.80	65.57
1	6	03 26.75	04 35.80	74.96
1	7	03 52.66	04 35.80	84.36
1	8	04 18.59	04 35.80	93.76
2	1	01 43.20	04 36.24	37.36
2	2	02 09.20	04 36.24	46.77
2	3	02 33.16	04 36.24	55.44
2	4	02 58.14	04 36.24	64.49
2	5	03 22.97	04 36.24	73.48
2	6	03 47.99	04 36.24	82.53
2	7	04 12.90	04 36.24	91.55
3	1	02 08.55	04 37.09	46.40
3	2	02 34.57	04 37.09	55.78
3	3	02 59.73	04 37.09	64.86
3	4	03 25.58	04 37.09	74.19
3	5	03 51.59	04 37.09	83.58
3	6	04 17.31	04 37.09	92.86
4	1	02 33.61	04 37.51	55.35

TABLE 21
THE INSINP CASE (CONT'D.)

input inserted	gates replaced	edit time min sec	compilation time min sec	ratio %
4	2	03 00.03	04 37.51	64.87
4	3	03 26.26	04 37.51	74.33
4	4	03 52.92	04 37.51	83.93
4	5	04 19.81	04 37.51	93.62
5	1	03 00.15	04 37.99	64.80
5	2	03 26.38	04 37.99	74.24
5	3	03 53.49	04 37.99	83.99
5	4	04 20.42	04 37.99	93.68
6	1	03 27.49	04 38.48	74.51
6	2	03 55.00	04 38.48	84.39
6	3	04 12.21	04 38.48	90.57
7	2	04 23.65	04 39.04	94.48

inserted, and the number of gates being replaced due to those insertions. The measurement of the performance of the program in this case was done by comparing the amount of time needed for the Editor program to modify the circuit and the time needed for the recompilation of the circuit.

Table 23 presents the results of the tests for the ALU circuit. The table is basically the same of the insert input case, with the insertion of one more column for the presentation of the number of gates inserted in the circuit.

By the results presented in the table we can see that the Editor program allows an insertion of a great number of inputs and gates in the circuit. The tests that were done did not cover all the possible insertions that could be done in the circuit in a time lower than the time necessary for recompilation of the circuit, but we can see by the results that we are modifying a number of variables that is almost equal to 6% of the number of gates in the circuit..

TABLE 22

THE INSINP CASE
FOR THE TEST CIRCUIT

input inserted	gates replaced	edit time min sec	compilation time min sec	ratio %
1	1	00 24.35	00 43.13	56.46
1	2	00 27.83	00 43.13	64.53
1	3	00 31.52	00 43.13	73.08
1	4	00 35.33	00 43.13	81.92
1	5	00 39.18	00 43.13	90.84
1	6	00 43.08	00 43.13	99.88
2	1	00 28.78	00 44.03	65.36
2	2	00 30.82	00 44.03	70.00
2	3	00 35.28	00 44.03	80.13
2	4	00 39.43	00 44.03	89.55
2	5	00 43.64	00 44.03	99.11
3	1	00 33.21	00 44.98	73.83
3	2	00 37.39	00 44.98	83.13
3	3	00 41.84	00 44.98	93.02
4	1	00 37.83	00 45.90	82.42
4	2	00 42.25	00 45.90	92.05
5	1	00 42.68	00 46.82	91.16

Table 24 presents the results obtained in the INSINPG case for the TEST circuit. The table has the same format as the ALU table.

In this case only few tests were done in the circuit. Even with a small number of tests we can see that the system can increase the number of inputs and gates inserted in the same time that was need for recompilation of the circuit.

TABLE 23

THE INSINPG CASE
FOR THE ALU CIRCUIT

input inserted	gates inserted	gates replaced	edit time min sec	compil. time min sec	ratio %
1	1	1	01 50.78	04 39.74	39.60
1	1	2	02 15.65	04 39.74	48.49
1	1	3	02 40.40	04 39.74	57.34
1	1	4	03 05.52	04 39.74	66.32
1	1	5	03 30.97	04 39.74	75.42
1	1	6	03 56.29	04 39.74	84.47
1	1	7	04 21.47	04 39.74	93.47
1	2	1	02 21.19	04 42.08	50.05
1	2	2	02 47.79	04 42.08	59.48
1	2	3	03 13.83	04 42.08	68.71
1	2	4	03 39.82	04 42.08	77.93
1	2	5	04 05.94	04 42.08	87.19
1	2	6	04 31.96	04 42.08	96.41
1	3	1	02 53.96	04 44.52	61.14
1	3	2	03 21.45	04 44.52	70.80
1	3	3	03 47.12	04 44.52	79.83
1	3	4	04 12.70	04 44.52	88.82
1	3	5	04 38.01	04 44.52	97.71
1	4	1	03 29.08	04 45.80	73.16
1	4	2	03 55.23	04 45.80	81.61
1	4	3	04 21.25	04 45.80	91.41
1	5	1	04 05.23	04 47.14	85.40
1	5	2	04 31.94	04 47.14	94.71

TABLE 23
THE INSINPG CASE (CONT'D.)

input inserted	gates inserted	gates replaced	edit time min sec	compil. time min sec	ratio %
1	6	1	04 41.94	04 48.62	97.69
2	1	1	02 16.82	04 40.39	48.80
2	1	2	02 40.77	04 40.39	57.34
2	1	3	03 06.44	04 40.39	66.49
2	1	4	03 32.26	04 40.39	75.70
2	1	5	03 58.14	04 40.39	84.93
2	1	6	04 23.94	04 40.39	94.13
2	2	1	02 46.68	04 42.72	58.96
2	2	2	03 13.56	04 42.72	68.46
2	2	3	03 39.04	04 42.72	77.48
2	2	4	04 05.26	04 42.72	86.75
2	2	5	04 32.11	04 42.72	96.25
2	3	1	03 16.83	04 45.15	69.03
2	3	2	03 43.04	04 45.15	78.22
2	3	3	04 02.31	04 45.15	84.98
2	3	4	04 31.87	04 45.15	95.34
3	1	1	02 34.97	04 41.07	55.14
4	1	1	02 52.08	04 41.72	61.08
5	1	1	03 10.47	04 42.40	67.45

M. THE DELINP CASE

In this case two parameters are important in the measurement of the performance of the Editor program: how many inputs are being deleted and how many gates will be replaced due to those deletions. The measurement of the performance of

TABLE 24

THE INSINPG CASE
FOR THE TEST CIRCUIT

input inserted	gates inserted	gates replaced	edit time min sec	compil. time min sec	ratio %
1	1	1	00 29.57	00 45.22	65.39
1	1	2	00 32.30	00 45.22	71.43
1	1	3	00 34.85	00 45.22	77.07
1	1	4	00 37.42	00 45.22	82.75
1	1	5	00 40.15	00 45.22	88.79
1	1	6	00 42.85	00 45.22	94.76
1	2	1	00 34.89	00 45.85	76.10
1	2	2	00 38.11	00 45.85	83.12
1	2	3	00 41.08	00 45.85	89.60
1	2	4	00 44.13	00 45.85	96.25
1	3	1	00 39.21	00 46.17	84.93
1	3	2	00 42.17	00 46.17	91.34
1	3	3	00 45.32	00 46.17	98.16
1	4	1	00 44.61	00 46.55	95.83
2	1	1	00 32.78	00 46.16	71.01
2	1	2	00 35.42	00 46.16	76.73
2	1	3	00 38.09	00 46.16	82.52
2	1	4	00 41.28	00 46.16	89.43
2	1	5	00 44.96	00 46.16	97.40
2	2	1	00 38.21	00 46.81	81.63
2	2	2	00 41.19	00 46.81	87.99
2	2	3	00 44.65	00 46.81	95.39
2	3	1	00 43.02	00 47.99	89.64

TABLE 24
THE INSINPG CASE (CONT'D.)

input inserted	gates inserted	gates replaced	edit time min sec	compil. time min sec	ratio %
2	3	2	00 46.57	00 47.99	97.04
3	1	1	00 36.15	00 46.91	77.06
4	1	1	00 41.25	00 47.65	86.57

the program in this case was done by comparing the amount of time needed by the Editor program to modify the circuit and the time needed by the recompilation of the circuit.

Table 25 presents the result of the tests for the ALU circuit. The table is basically the same of the insert case with the gates inserted column replaced by the inputs deleted column.

As we can see the Editor program allows that 1 input be deleted and 8 gates be replaced or 7 inputs be deleted and 2 gates be replaced in a time lower than the time needed by the compiler to recompile the entire circuit.

Table 26 presents the result of the tests for the TEST circuit. The table format is the same of the ALU case.

In this case the Editor program allows 1 input to be deleted and 5 gates to be modified or 3 inputs to be deleted and 2 gates to be modified in an amount of time lower than the time needed to recompile the entire circuit. This is a reasonable result because we are modifying almost 35% of the number of gates in the circuit in a time lower than the time to recompile the entire circuit.

N. THE DELINPG CASE

In this case three parameters are important in the measurement of the performance of the Editor program: the number of inputs being deleted, the number of gates being deleted, and the number of gates to be replaced due to those deletions. The measurement of the performance of the program in this case was done by comparing the amount of time needed by the Editor program to modify the circuit and the time needed for the recompilation of the circuit.

TABLE 25

THE DELINP CASE
FOR THE ALU CIRCUIT

input deleted	gates replaced	edit time min sec	compilation time min sec	ratio %
1	1	01 18.33	04 35.21	28.46
1	2	01 41.62	04 35.21	36.92
1	3	02 08.21	04 35.21	46.51
1	4	02 33.77	04 35.21	55.87
1	5	02 59.61	04 35.21	65.26
1	6	03 25.17	04 35.21	74.55
1	7	03 51.93	04 35.21	84.27
1	8	04 17.72	04 35.21	93.64
2	1	01 43.47	04 34.83	37.65
2	2	02 07.92	04 34.83	46.55
2	3	02 34.92	04 34.83	56.37
2	4	03 01.31	04 34.83	65.97
2	5	03 26.99	04 34.83	75.32
2	6	03 51.33	04 34.83	84.17
2	7	04 16.46	04 34.83	93.32
3	1	02 07.84	04 33.98	46.66
3	2	02 33.15	04 33.98	55.90
3	3	03 00.01	04 33.98	65.70
3	4	03 26.16	04 33.98	75.25
3	5	03 52.79	04 33.98	84.97
3	6	04 15.12	04 33.98	93.12
4	1	02 35.16	04 33.11	56.81

TABLE 25
THE DELINP CASE (CONT'D.)

input deleted	gates replaced	edit time min sec	compilation time min sec	ratio %
4	2	03 01.80	04 33.11	66.57
4	3	03 27.45	04 33.11	75.96
4	4	03 54.02	04 33.11	85.69
4	5	04 20.53	04 33.11	95.39
5	1	02 59.12	04 32.23	65.80
5	2	03 26.45	04 32.23	75.84
5	3	03 53.01	04 32.23	85.59
5	4	04 19.16	04 32.23	95.20
6	1	03 25.33	04 30.97	75.78
6	2	03 51.72	04 30.97	85.52
6	3	04 18.71	04 30.97	95.48
7	2	04 21.69	04 29.75	97.01

Table 27 presents the result of the tests for the ALU circuit. One more column is added to allow the presentation of the number of replacements of the input (gate) inserted column(s) by the input (gate) deleted column(s).

By the results presented in the table we can see that the Editor program allows the deletion of a great number of inputs and gates in the circuit. The tests that were done did not cover all the possible deletions that could be done in the circuit in a time lower than the time necessary for recompilation of the circuit, but we can see by the results that we are modifying a number of variables that is almost equal to 6% of the number of gates of the circuit.

Table 28 presents the results obtained in the delete input and gate case for the TEST circuit. The table has the same format as the ALU table.

TABLE 26

THE DELINP CASE
FOR THE TEST CIRCUIT

input deleted	gates replaced	edit time min sec	compilation time min sec	ratio %
1	1	00 25.16	00 42.36	59.40
1	2	00 29.78	00 42.36	70.30
1	3	00 32.67	00 42.36	77.12
1	4	00 35.98	00 42.36	84.94
1	5	00 40.51	00 42.36	95.63
2	1	00 30.43	00 41.85	72.71
2	2	00 34.99	00 41.85	83.61
2	3	00 39.00	00 41.85	93.19
3	1	00 35.23	00 40.12	87.81
3	2	00 39.89	00 40.12	99.43

Also in this case some of the tests were not done, to decrease the number of tests that were applied to the circuit. Even with a small number of tests we can see that the system can delete a number of inputs and gates almost equal to 40% of the gates of the circuit in less time than the time needed for the compilation of the entire circuit.

TABLE 27

THE DELINPG CASE
FOR THE ALU CIRCUIT

input deleted	gates deleted	gates replaced	edit time min sec	compil. time min sec	ratio %
1	1	1	01 58.67	04 33.91	43.32
1	1	2	02 22.03	04 33.91	51.85
1	1	3	02 49.03	04 33.91	61.71
1	1	4	03 14.45	04 33.91	70.99
1	1	5	03 39.72	04 33.91	80.22
1	1	6	04 05.33	04 33.91	89.57
1	1	7	04 30.86	04 33.91	98.89
1	2	1	02 46.31	04 32.02	61.14
1	2	2	03 01.13	04 32.02	66.59
1	2	3	03 25.72	04 32.02	75.63
1	2	4	03 50.21	04 32.02	84.63
1	2	5	04 16.73	04 32.02	94.38
1	3	1	02 56.12	04 30.87	65.02
1	3	2	03 21.87	04 30.87	74.53
1	3	3	03 47.89	04 30.87	84.13
1	3	4	04 13.05	04 30.87	93.42
1	4	1	03 29.31	04 28.71	77.89
1	4	2	03 56.11	04 28.71	87.87
1	4	3	04 22.42	04 28.71	97.66
1	5	1	04 06.03	04 27.67	91.92
2	1	1	02 16.35	04 32.83	49.98
2	1	2	02 41.26	04 32.83	59.11
2	1	3	03 06.99	04 32.83	68.54

TABLE 27

THE DELINPG CASE (CONT'D.)

input deleted	gates deleted	gates replaced	edit time min sec	compil. time min sec	ratio %
2	1	4	03 33.08	04 32.83	78.10
2	1	5	03 59.23	04 32.83	87.68
2	1	6	04 25.21	04 32.83	97.21
2	2	1	02 49.24	04 30.85	62.48
2	2	2	03 13.51	04 30.85	71.45
2	2	3	03 40.01	04 30.85	81.23
2	2	4	04 06.41	04 30.85	90.98
2	3	1	03 17.51	04 29.38	73.32
2	3	2	03 44.65	04 29.38	83.40
2	3	3	04 03.93	04 29.38	90.55
3	1	1	02 36.49	04 30.72	57.81
4	1	1	02 54.05	04 29.29	64.63
5	1	1	03 11.93	04 27.94	71.63

VI. RECOMMENDATIONS, FURTHER WORK AND CONCLUSIONS

A. RECOMMENDATIONS

The Editor program and the compiler program were written in a way that tries to make them as general as possible. However, due to the possibility that the number of gates supported by the system might be increased, some attention need to be given to the maintenance of both programs. Also the user needs to remember some points when trying to use the compiler or editor program to simulate some digital circuit.

The first point to be remembered is about the usage of submodules in the description of the circuit. When the user defines a circuit by using submodules, the compiler program takes those submodules and expands them into lower levels to allow the usage of gates that are already described in the library of the system. When the system does this expansion it loses track of the submodules that were defined by the user. If, after the compilation, the user tries to edit this circuit he/she cannot use the submodules that were described before because the program will not be able to find them. Instead, the user needs to define the modifications by using the definition of gates in the system after the expansion. To do the description in this way the user needs to verify how each variable was defined by using the SYMT table, DESCT table and INI table that were built by the system and saved in the user disk as files with the extensions STB, DCT and INI, respectively.

Another point where the user needs to be careful when using the MultiSim package is the insertion of gates or user defined primitives in the library of the system. One of the enhancements done in the system by Kelly was the insertion of the ADSTRUC command in the system([Ref. 8]). This command will allow the user to insert any circuit in the library as a user defined circuit, but this insertion will not allow the use of this circuit as a primitive from the library in the Editor program, because the only part that was added to the system was the structural description of the circuit. In this case when the user uses the new circuit in the description of a higher level circuit, the compiler will not use the inserted circuit as a primitive and will expand it, to use the characteristics of the lower level circuits that compose it. Because of this expansion the same problem that was described in the previous paragraph will happen, and the system will loose track of this new primitive, and the user cannot use it in the Editor

program. The only possibility for the use of a user defined primitive in the system without any restriction is the ADDLIB command (not implemented yet), that inserts not only the structural description but also the block description of the circuit, and this is the way that the primitives need to be described to be used without problems.

Because, until now, all the primitives that were defined in the system had 6 or less inputs the programs were prepared to work with gates (or circuits) that have up to 10 inputs. To allow for a gate with a greater number of inputs to be supported by the system, the following modifications need to be made to the programs:

1. In all the programs the description of the structure of the INI table needs to be modified. This definition appears in the declaration STRUCT INP_NAME and the modification that needs to be done is the insertion of the room to allow a greater number of inputs in the table. This insertion needs to be done after the INP10[8] part of the description by putting the declaration CHAR INP11[8], INP12[8], until there is room for all the inputs be placed in the table.
2. Since a great number of inputs could appear in the table the CKT subroutine in the CADD program (Appendix A) needs to be modified to work with the new number of inputs. This modification needs to be done in the part of the subroutine that fills the places of the INI table when the compiler reads the description of the gate.
3. In the compiler program and in the editor program the segments of code that copy the INI table to a file and vice-versa need to be modified to allow a copy of a table with more elements. Those parts are in the end of the main routine in the compiler program (Appendix A) and in the beginning and in the end of the main routine of the editor program (Appendix D).
4. The MDFYFAN subroutine of the editor program (Appendix D) needs to be modified to allow the update of the fanload when the system works with gates with more than 10 inputs.
5. The DELGATE subroutine of the editor program (Appendix D) needs to be modified to allow the deletion and use of gates with more than 6 inputs.
6. The IDINP subroutine of the PRCMP1 routines (Appendix E) needs to be modified for the same reason that the CKT subroutine was modified in the compiler program.

The points that were presented in this section are very important because if they were not followed as presented the system will have problems in the future.

B. FURTHER WORK

Even with the great improvements done to the MultiSimPC package with the insertion of the Editor program, there are other improvements which could be made in the system.

The first improvement that needs to be made is the communications between the user and the MultiSimPC. At this stage we cannot consider the system to be user friendly, since the user needs to write two or more files, depending of the work that he/she wants to do and also the user needs to know all the VOHL syntax to describe the circuits for compilation and editing. To improve the way that the system will be used the system should be made into an interactive program, where the user calls a program and the program asks him/her about the possible things that can be done with the program and the user selects the segment that he/she wants. All the information that the computer needs to perform the job will be asked of the user by the computer. This is not a very difficult problem and was not implemented only because of the time required for preparation of this thesis. However, this could be a great improvement in the system because the user will not need to know the syntax and the meaning of the several commands and will no longer need to write the files necessary for the implementation of the circuit.

Another improvement that can be done in the system is the insertion of the command ADDLIB in the system. As was explained before, the insertion of this command will allow the insertion of user defined primitives in the library of the system without any restriction with respect to the Editor program. In the [Ref. 9] Kelly presented all the directives to the insertion for the ADBLOCK command that will insert the block description of the primitive in the system. If the ADBLOCK command was implemented, the operations that are performed in it can be joined with the operations done in the ADSTRUC command to generate the ADLIB command.

Another point that needs to be considered in the system is the design of a graphic capability to take the circuits presented in the VOHL syntax and display them in graphic form in the screen. This will certainly be a very difficult challenge to execute but when ready it will greatly improve the versatility of the MultiSimPC package. A possible solution for this improvement was presented by Kelly in the ([Ref. 9]).

C. CONCLUSIONS

The results of the tests performed on the different circuits shows that the Editor program could be a very good help in the debugging of digital circuits. As we can see in the various tables presented in the last chapter, the program had good results in each of the cases that it was designed to work with.

The results presented in that chapter can not be taken as a base for the time that the system expends to modify a circuit with a specific number of gates. In general we

cannot say that this program will not be advantageous to replace 10 gates in a circuit with 142 gates just because in the ALU circuit the time to do this replacement with the editor program was greater than the time needed for recompilation. This time will vary from one circuit to another as a function of the different characteristics that are inserted in the circuit. Even the same circuit could produce significant differences between the times for edition. Suppose that we have a circuit with 142 gates where 130 of them are already initialized. Suppose now that we want to replace a gate in this circuit, but the output of this gate was not initialized. In this case the system will not need to work with the initialization file, since that variable did not appear in it, and consequently it will need less time than the case where we want to replace a gate that is already initialized and will need more work to complete the command.

The tests that were realized show the correct time for edition in the ALU and TEST circuits only when they present the characteristics (initial values, modified delays and printouts) that appear in the original description of the circuits (Appendix H and Figure 1.2, respectively). A modification in any of those characteristics can drastically modify both the time for compilation and the time for the edition of the circuit.

However, the important point is that the Editor program has the goal of help the debugging of the design of digital circuits. After a circuit is designed and tested the normal debugging is done by making small changes in the circuit and verifying the effects of those changes in the behavior of the circuit. In this way the Editor program fulfil its function, because for small changes it always has a better performance than the compiler program.

```
int DEF() ;                 /*   DEFINE parsing routine        */
int INI() ;                 /*   INITIALIZE parsing routine    */
int PRI() ;                 /*   PRINTOUT parsing routine      */
extern int add_lib();       /*   adds primitives to library    */
int rfdel() ;               /*   rise/fall delay handling      */
int bdread() ;              /*   block delay reading routine   */
int cmode() ;               /*   code generation for mode      */
int matgen() ;              /*   delay matrix and mode gen.    */
int parseid() ;             /*   single id parsing             */
int findid() ;              /*   finds symbol table index      */
int finddesc() ;            /*   finds symbol table index for  */
                            /*   the given function name/type  */
int updesct() ;             /*   updates the descriptor table  */
                            /*   (name and symbol table index)*/
int findprim() ;            /*   finds primitive library index*/
int update() ;              /*   update symbol table           */
int connect() ;             /*   code gen. and fanld update    */
                            /*   (descriptor interconnections)*/
int code_input() ;          /*   code generation for INPUTS    */
                            /*   declaration                   */
int errmessage() ;          /*   error message printing        */
int outerror() ;            /*   output error message          */
int error() ;               /*   error handling routine        */
int firstp() ;              /*   first pass for expand         */
int secondp() ;             /*   second pass for expand        */
int pdelim() ;              /*   prints a file of keywords and */
                            /*   tokens, separated by delimiters*/
int ftype() ;               /*   finds type of a identifier    */
int reverse() ;             /*   reverses a string             */
int fcopy() ;               /*   file copying routine          */
int copy_noend();           /*   copies everything but END token */
int itoa() ;                /*   integer to ASCII routine      */
int cexpand() ;             /*   expands the primitive in ckt */
int substitute() ;          /*   substitutes the func's code  */
int foutorder() ;           /*   one to one correspondence for*/
int finorder() ;            /*   actual and formal parameters */
int ckt_line() ;            /*   scans one line of ckt desc.  */
int search() ;              /*   search a delimiter or toknn  */
int tack() ;                /*   tacks a number to an id       */
int multi_mod() ;           /*   sub module handling routine  */
int fadvance() ;            /*   advances to next line         */
int hashf() ;               /*   converts input name to number*/
int test();                 /*   tests for hash collisions     */
/*-------------------------------------------------------------*/


/*----------------DATA STRUCTURES-----------------------------------*/
struct sym_tab {            /*  symbol table                        */
  char name[8] ;            /*  name = name of id                   */
  int descno, funcno ;      /*  descno = descriptor number          */
                            /*  funcno = primitive lib index        */
  int fanld;                /*  fanld = actual circuit load         */
  int despos, delpos;       /*  despos = descriptor spaces on file  */
                            /*  delpos = delay spaces on file       */
  int ini_num, pri_num;     /*  ini_num = initialization order      */
                            /*  pri_num = printout order            */
  int pri_val;              /*  pri_val = # characters in variable  */
 } ;

struct desc_tab {           /*  table containing function           */
  char fun[8] ;             /*  names (type names) and their        */
  int dnum ;                /*  symbol table indexes                */
 } ;

struct norm_tab {           /*  table containing function           */
  char nom[8] ;             /*  names/types and associated          */
  int nmld ;                /*  normload declared in DEFINE         */
 } ;

struct prim_tab {           /*  Primitive table :                   */
```

```
  char nam[8] ;              /*  primitive table                      */
  char nam2[8] ;             /*  EXTENDED primitive table             */
  int numpar, outp ;         /*  numpar = no. of parameters           */
  int normld, fanout ;       /*  for the function                     */
  int technology, overld ; /*  outp = # of outputs                    */
 } ;

struct err_stack {           /* stack for errors in one line          */
  char nm[8] ;               /* nm = name of unexpected id            */
  int errno ;                /* errno = error number                  */
 } ;

struct exp_tab {             /* expand table                          */
  char fname[8] ;            /* fnum = prim lib index                 */
  int fnum ;
 } ;

struct namepair {            /* this holds system-generated           */
  char e_from[8];            /* expansion requests                    */
  char e_to[8];
 };

struct swapname {       /* each of these nodes will contain a module */
  char sname[8];             /* specified in the USING parameter list */
  int used;                  /* this indicates whether used or not    */
  };

struct inp_name {            /* holds all the inputs for each gate    */
  char iname[8];             /* iname = name of the variable          */
  int inp_num;               /* inp_num = # of inputs in the gate     */
  char inp1[8], inp2[8];     /* inp1 to inp10 = inputs for the gate   */
  char inp3[8], inp4[8];
  char inp5[8], inp6[8];
  char inp7[8], inp8[8];
  char inp9[8], inp10[8];
  int ifin;                  /* ifin = termination of the table       */
  };

struct tab_del               /* holds all the information about the   */
  {                          /* gates that have modified delays       */
  int indx, dsc_nb;          /* indx = index of the table             */
                             /* dsc_nb = descriptor number            */
  int typ_num, num_ipt;      /* typ_ num = type of modification       */
                             /* num_ipt = gate's input to be modified */
  int num_opt, val;          /* num_opt = gate's output to be modified*/
                             /* val = spaces occupied in the file     */
  };
/*-----------------------------------------------------------------------*/

/*---------------STORAGE ALLOCATION-----------------------------------*/
struct sym_tab symt[maxsym] ;
struct desc_tab desct[maxsym] ;
struct norm_tab nort[maxnorm] ;
struct prim_tab primt[maxprim], *primptr ;
struct err_stack errt[5] ; /* max of 5 errors per line     */
struct exp_tab expt[30] ;  /* max 30 expansion requests    */
struct swapname typelist[maxprim][8];  /* table of module replacements*/
struct swapname swaplist[20][8];  /* USING parameter list storage */
struct namepair reqtable[maxprim];
struct inp_name inptab[maxsym];
struct tab_del del_tab[100];
struct sym_tab temporary ;

int req_count;               /* number of system expand reqs */
int line_count;              /* line index for swaplist      */
int swap_flag;               /* indicates whether all of this*/
                             /* is necessary or not          */
extern int found_start;      /* marks occurrance of SWAPLIN  */
extern int found_end;        /* marks occurrance of ENDSWAP  */
int sfound;
```

```
int add_flag;                /* set if cell is to be added to*/
                             /* primitive library            */
int err_ptr ;                /* error table pointer (count)  */
                             /* for one line.                */
int matcount ;               /* delay matrix count           */
int delimiter, bb, skip ;    /* delimiter = delimiter type   */
                             /* bb = buff[80] index (line)   */
int rdmat[maxouts][maxouts], fdmat[maxouts][maxouts] ;
                             /* rise and fall delay matrices */
int toknn, err_count ;       /* err_count  = error count     */
int desc_no , sym_count, symid ;   /* desc_no = desc. count*/
int dptr, descid, lim ;      /* descriptor table indices     */
                             /* dptr = descriptor table cnt  */
int normcount ;              /* normtable count              */
int expcount ;               /* expand table count           */
int cellcount;               /* # of MODULEs in user program */
int filecount,pct;           /* poutcount used for debugging */
int prinpflag;               /* controls printing of source  */
int print_select;            /* determines whether a ')' causes*/
                             /* a linefeed or not (default=no)*/
int prim_count, primid ;     /* prim_count = # of primitives */
int sys_prims ;              /* number of permanent (system) */
                             /* primitives                   */
int savprim ;
int inpcount,outpcount;      /* number of inputs and outputs */
                             /*(used to add a new primitive) */
int features[maxprim][2];    /* first field describes the type of */
                             /* descriptions available;          */
                             /* -1 -> empty     0 -> block only  */
                             /* 1 -> struc only  2 ->block & struc */
                             /* second field indicates primitive level */
int append_table[maxprim]; /* keeps track of the functions we've */
int append_index;            /* added to the user program        */
int expdone;                 /* marks completion of struc expansion*/
int no_compilation;          /* used when only making library additions*/
char token_buf[8], savbuf[8], buff[80] ; /* buff = 1 line  */
char keyword[maxkey][8] ;  /* keyword table                  */
char inch;
char instack[maxouts][8], outstack[maxouts][8] ;
char in1stack[maxouts][8], out1stack[maxouts][8] ;
char in2stack[maxouts][8], out2stack[maxouts][8] ;
/* in/out stack = inputs/outputs from primitive's definition in  */
/* library. in1/out1 = calling inputs/outputs (user program)     */
/* in2/out2 = inputs/outputs of each line in primitive's desc.   */

char typstack[maxouts][8] ;/* types of primitive to be expanded  */
int typcount ;
int index, act_val ;
int index1, val_prt ;
int ord_ini, ord_pri;
int hashtable[100];            /* the hash table                 */
int hashcount;                 /* number of items in hashtable   */
int inum;
int valact ;
int incount , outcount ;
int in1count, out1count, in2count, out2count ;
int outorder, inorder ;
int count ;              /* for printing on the file */
char savfunc[8], userprg[8] ;
int ft, occurance ;      /* occurance = # of times call to    */
                         /* primitive to be expanded is made  */

FILE *r1 ;               /* pointer to input data file        */
FILE *r2 ;               /* pointer to STRUCT library         */
FILE *r3 ;               /* pointer to modified STRUC library */
FILE *r4 ;               /* pointer to user STRUC description */
FILE *w1 ;               /* pointer to expanded file   P1EXP  */
FILE *t1 ;               /* pointer to temp file              */
FILE *li1 ;              /* pointer to library                */
FILE *s1 ;               /* pointer to primitive's desc. SCR1 */
```

```
FILE *s2 ;              /* pointer to expanded circuit  SCR2 */
FILE *rp ;              /* read pointer to input data file   */
FILE *wq ;              /* circuit description for simulator */
FILE *sy ;              /* symt table stored for edition     */
FILE *de ;              /* desct table stored for edition    */
FILE *nm ;              /* nort table stored for edition     */
FILE *ip ;              /* input table stored for edition    */
FILE *dp ;              /* delay table of the descriptors    */
FILE *tm ;              /* modified delay file for simulation*/
FILE *ti ;              /* initialization file for simulation*/
FILE *tc ;              /* descriptor file for simulation    */
FILE *td ;              /* default delay file for simulation */
FILE *tp ;              /* printout file for simulation      */
/*-----------------------------------------------------------------*/

/*******************************************************************
 *                                                                 *
 *                          MAIN PROGRAM                           *
 *                                                                 *
 *******************************************************************/

main(argc, argv)
int argc ;
char *argv[];
 {
  int i, j;
  strcpy(userprg,argv[1]);
  prinpflag = 0 ;
  filecount = 0 ;
  inpcount=0;
  outpcount=0;
  no_compilation=0;             /* always assume we're compiling */
  print_select=0;               /* print ')' without a linefeed  */
  req_count=0;
  swap_flag=0;
  line_count=0;
  hashcount=0;
  index = 0;
  index1 = 0 ;
  for (i=1; i<100; i++)
     hashtable[i] = -1;

/*----------------------PRIMITIVES SUPPORTED---------------------*/
  for (i = 0; i < maxprim; i = i + 1)
   {
    primt[i].normld = 1 ;
    primt[i].fanout = 20 ;
    primt[i].technology = 0 ;
    primt[i].overld = 5 ;
   }

   primsetup(&primt[0]);            /* initialize primitives */
   sys_prims=prim_count;            /* primcount may change, but */
                                    /* we need a copy of its     */
                                    /* starting value            */

/*--------------------INITIALIZE-------------------------------*/
                          /* initialize SYMT table          */
 for (i = 0; i < maxsym; i = i + 1)
  {
   symt[i].fanld = 0 ;
   symt[i].descno = -1 ;
   symt[i].funcno = -1 ;
   symt[i].despos = 0 ;
   symt[i].delpos = 0 ;
   symt[i].ini_num = 0 ;
   symt[i].pri_num = 0 ;
   symt[i].pri_val = 0 ;
  }
                         /* initialize IPT  table          */
```

```
 for (i = 0; i < maxsym; i++)
  {
     strcpy(inptab[i].inp1,"xxx") ;
     strcpy(inptab[i].inp2,"xxx") ;
     strcpy(inptab[i].inp3,"xxx") ;
     strcpy(inptab[i].inp4,"xxx") ;
     strcpy(inptab[i].inp5,"xxx") ;
     strcpy(inptab[i].inp6,"xxx") ;
     strcpy(inptab[i].inp7,"xxx") ;
     strcpy(inptab[i].inp8,"xxx") ;
     strcpy(inptab[i].inp9,"xxx") ;
     strcpy(inptab[i].inp10,"xxx") ;
  }
 for (i = 0; i < 20 ; i = i + 1)
  expt[i].fnum = -1 ;
 for (i=0; i<20; i++)
    {
    for (j=0; j<8; j++)
       {
       swaplist[i][j].used=0;           /* set USING parameter lists */
       }                                /* to EMPTY                  */
    }
 for (i=0; i<maxprim; i++)
    {
    for (j=0; j<8; j++)
       {
       typelist[i][j].used=0;           /* set type list to empty */
       }
    }
/*-----------------------------------------------------------------*/

/*---------------------KEYWORDS------------------------------------*/
 strcpy(keyword[0], "MODULE") ;
 strcpy(keyword[1], "INPUTS") ;
 strcpy(keyword[2], "OUTPUTS") ;
 strcpy(keyword[3], "TYPES") ;
 strcpy(keyword[4], "{") ;
 strcpy(keyword[5], "}") ;
 strcpy(keyword[6], "INITIAL") ;
 strcpy(keyword[7], "PRINTOU") ;
 strcpy(keyword[8], "INTERNA") ;
 strcpy(keyword[9], "DEFINE") ;
 strcpy(keyword[10],"RISEDEL") ;
 strcpy(keyword[11],"FALLDEL") ;
 strcpy(keyword[12],"TECHNOL") ;
 strcpy(keyword[13],"TTL") ;
 strcpy(keyword[14],"NMOS") ;
 strcpy(keyword[15],"CMOS") ;
 strcpy(keyword[16],"ECL") ;
 strcpy(keyword[17],"FANOUT") ;
 strcpy(keyword[18],"NORMLOA") ;
 strcpy(keyword[19],"OVERLOA") ;
 strcpy(keyword[20],"END") ;
 strcpy(keyword[21],"EXPAND") ;
 strcpy(keyword[22],"USING");              /* used to replace modules*/
 strcpy(keyword[23],"SWAPLIN");            /* marks each ckt line  */
 strcpy(keyword[24],"ENDSWAP");            /* to be examined for swaps*/
 strcpy(keyword[25],"NOEXP");
 strcpy(keyword[26],"ADDLIB");             /* add a cell primitive */
 strcpy(keyword[27],"ADSTRUC");            /* struc-only description*/
 strcpy(keyword[28],"ADBLOCK");            /* block-only description*/
/*-----------------------------------------------------------------*/

  err_ptr = -1 ;          /* error count for one line   */
  err_count = 0 ;               /* error count for program    */
  expcount = 0 ;
  printf("Opening the circuit descriptor file... \n");
  for (i=0; i<maxprim; i++)              /* initialize the append table */
     append_table[i]=-1;                 /* to empty                    */
  append_index=0;
```

```
cellcount=0;
r1=fopen(argv[1],"r");
countcells(r1);                          /* count the # of MODULEs  */
fclose(r1);
r1=fopen(argv[1],"r");
r2=fopen("d:outfile","w");              /* the "stripped" file       */
build();                                 /* strip off the END keyword */
fprintf(r2," END; \n");
fclose(r2);
if ((no_compilation==1) && (add_flag==1))  /* no_compilation set# */
   {
   add_lib();                  /* yes, add the modules without compiling*/
   }
else
   {                                     /* no, begin compilation */
   r1=fopen("d:outfile","r");
   w1=fopen("d:p11","w");                 /* copy user prog to "p11"  */
   fcopy(r1,w1);                          /* (now we're back to       */
   fclose(r1);                            /* Ausif's code)            */
   fclose(w1);
   printf("Files are restored.  Multimodule expansion begins.\n");
   /*-----------------------EXPANSION-----------------------------*/
   firstp() ;   /* first pass, determine any expansion requests,  */
                /* put each request on expand table (expt)        */
   perform_expansion();      /* any expansions handled in here    */
   r1=fopen("d:p11","r");           /* copy p11 to INFILE */
   r2=fopen("d:infile","w");
   fcopy(r1,r2);
   fclose(r1);
   fprintf(r2, "END; \n");
   fclose(r2);
   r1=fopen("d:infile","r");        /* copy INFILE to OUTFILE */
   r2=fopen("d:outfile","w");       /* but leave OUTFILE open */
   copy_noend(r1,r2);
   fclose(r1);
   cellcount=0;
   r1=fopen("d:infile","r");
   countcells(r1);                  /* count the # of MODULEs  */
   fclose(r1);
   r1=fopen("d:infile","r");
   struc_expand();                  /* handle any struc-only prims */
   r1=fopen("d:outfile","r");       /* copy this file back to p11 */
   r2=fopen("d:p11","w");           /* and expand the struc-only prims */
   fcopy(r1,r2);
   fclose(r1);
   fclose(r2);
   for (i=0; i<expcount; i++)
      expt[i].fnum = -1;            /* clear the expand table */
   expcount=0;
   firstp();                        /* and take care of any modules */
   if (expcount>0)                  /* that struc_expand() added    */
      perform_expansion();
   if (swap_flag==1)                /* any USINGS to deal with# */
      swap();                       /* if so, make the substitutions */

   /*-----------------end expansion-----------------------------*/
   rp = fopen("d:p11","r");          /* expanded user program */
   tm = fopen("d:modf","w");
   ti = fopen("d:initi","w");
   tc = fopen("d:descf","w");
   td = fopen("d:delf","w");
   tp = fopen("d:prnt","w");
                                     /* initialize compiler vars */
   prinpflag = 1 ;
   dptr = 0 ;                       /* desctable count           */
   normcount = 0 ;                  /* norm table count          */
   sym_count = 0 ;                  /* symble table entries count */
   inum = 0 ;
   desc_no = 0 ;                    /* descriptor count          */
   symid = -1 ;                     /* symbol table index        */
```

```
 matcount = 0 ;                          /* delay matrix count          */
 index = 0 ;
 index1 = 0 ;
 ord_pri = 1 ;
 ord_ini = 1 ;
 filecount = 0 ;
 act_val = 0 ;

 /*-----------------PARSING AND CODE GENERATION-------------------*/
 /* Recursive descent parsing is used . STD scheme is used for    */
 /* code generation. BNF is as follows.                           */
 /*                                                               */
 /* <COMPILE> => <MOD> <INP> <OUT> <TYP> { <CKT> } <DEF> <INI> <PRI>*/
 /* Non-terminals are defined in their respective sub-programs    */

 COMPILE() ;
 fclose (rp) ;
                                 /* save all tables      */
 sy = fopen ("d:symtable","w") ;
 fprintf(sy," ]d\n",sym_count);
 fprintf(sy," ]d ]d\n", ord_ini, ord_pri);
 for (i=0;i<sym_count;i++)
    {
      fprintf(sy," ]s ]d",symt[i].name,symt[i].descno);
      fprintf(sy," ]d ]d",symt[i].funcno,symt[i].fanld);
      fprintf(sy," ]d ]d",symt[i].despos,symt[i].delpos);
      fprintf(sy," ]d ]d",symt[i].ini_num,symt[i].pri_num);
      fprintf(sy," ]d\n",symt[i].pri_val);
    }
fclose(sy);
 dp = fopen ("d:deltable","w") ;
 fprintf(dp," ]d\n",index1);
 for (i=0;i<index;i++)
    {
      fprintf(dp," ]d ]d",del_tab[i].indx,del_tab[i].dsc_nb);
      fprintf(dp," ]d ]d",del_tab[i].typ_num,del_tab[i].num_ipt);
      fprintf(dp," ]d ]d\n",del_tab[i].num_opt,del_tab[i].val);
    }
fclose(dp);
de = fopen ("d:descptab","w");
fprintf(de," ]d\n",desc_no);
fprintf(de," ]d\n",dptr);
for (i=0;i<dptr;i++)
      fprintf(de," ]s ]d\n",desct[i].fun,desct[i].dnum);
fclose(de);
nm = fopen ("d:nortable","w") ;
fprintf(nm," ]d\n",normcount);
for (i=0;i<normcount;i++)
    {
      fprintf(nm," ]s ]d\n",nort[i].nom,nort[i].nmld);
    }
fclose(nm);
for (i = 0; i < inum; i++)
    inptab[i].ifin = 1 ;
 ip = fopen ("d:inptable","w") ;
 fprintf(ip," ]d\n",inum);
 for (i=0;i<inum;i++)
    {
      fprintf(ip," ]s ]d",inptab[i].iname,inptab[i].inp_num);
      fprintf(ip," ]s ]s",inptab[i].inp1,inptab[i].inp2);
      fprintf(ip," ]s ]s",inptab[i].inp3,inptab[i].inp4);
      fprintf(ip," ]s ]s",inptab[i].inp5,inptab[i].inp6);
      fprintf(ip," ]s ]s",inptab[i].inp7,inptab[i].inp8);
      fprintf(ip," ]s ]s",inptab[i].inp9,inptab[i].inp10);
      fprintf(ip," ]d\n",inptab[i].ifin);
    }
fclose(ip);
 if (err_count != 0)
    error(26) ;                 /* Compilation discontinued message */
 else
```

```
        error(38) ;                 /* no errors encountered message    */
      outerror() ;
      }
}

/*-----------------END OF MAIN PROGRAM--------------------------*/

/*****************************************************************
 *                                                               *
 *                    COMPILE SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/*---------------MAIN PARSING ROUTINE FOR 1 MODULE---------------*/
/* Recursive descent parser.  Syntax directed translation (SDT)  */
/* scheme is used for code generation.  BNF is as follows        */
/*                                                               */
/* <COMPILE> => <MOD> <INP> <OUT> <TYP> { <CKT> } <DEF> <INI> <PRI>*/
/* <--> = non terminals, all others are terminals                */

 COMPILE()
  {
   printf("Compilation begins.\n");
   MOD() ;                  /* call to MODULE parsing routine   */
   INP() ;                  /* call to INPUTS parsing routine   */
   OUT() ;                  /* call to OUTPUTS parsing routine  */
   skip = 0 ;
   TYP() ;                  /* call to TYPES parsing routine    */
   if (skip != 1)
    parseid(4) ;            /* '{' parsing                      */
   skip = 0 ;
   CKT() ;                  /* Circuit interconnections parsing */
                            /* '}' taken care of in CKT         */
   filecount = 0 ;
   matgen() ;               /* delay matrix generator           */
   filecount = 0 ;
   DEF() ;                  /* DEFINE parsing                   */
   filecount = 0 ;
   INI() ;                  /* INITIALIZE parsing               */
   filecount = 0 ;
   PRI() ;                  /* PRINTOUT parsing                 */
   filecount = 0 ;
   fprintf(tm," \n") ;      /* put a terminator on "modf"       */
   fprintf(ti," \n") ;      /* put a terminator on "initi"      */
   fprintf(tc," \n") ;      /* put a terminator on "descf"      */
   fprintf(td," \n") ;      /* put a terminator on "delf"       */
   fprintf(tp," 50\n") ;    /* put a terminator on "prnt"       */
   fclose(ti);              /* clean up and quit                */
   fclose(tc);              /* clean up and quit                */
   fclose(td);              /* clean up and quit                */
   fclose(tp);              /* clean up and quit                */
   fclose(tm);              /* clean up and quit                */
   if (add_flag==1)
      {
      add_lib();            /* perform additions to primitive library */
      }
  }
/*--------------------END COMPILE-------------------------------*/

/*****************************************************************
 *                                                               *
 *                      MOD SUBROUTINE                           *
 *                                                               *
 *****************************************************************/
/* <MOD> => MODULE <delimiter> id                                */

 MOD()
  {
   parseid(0) ;                 /* ADD and MODULE parsing  */
   parseid(maxkey) ;                /* module name             */
  }
```

```
/*----------------------END MOD-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                        INP SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* <INP> => INPUTS <delimiter> <IDSTRING>                        */

 INP()
  {
   parseid(1) ;                        /* INPUTS parsing         */
   IDSTRING(0) ;                       /* input names            */
  }
/*----------------------END INP-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                        OUT SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* <OUT> => OUTPUTS <delimiter> <IDSTRING>                       */

 OUT()
 {
  parseid(2) ;                /* OUTPUTS parsing                 */
  IDSTRING(-1) ;              /* -1 = outputs code for sym tab   */
 }
/*----------------------END OUT-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                        TYP SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* <TYP> => TYPES <delimiter> <T>                                */
/* <T> => <PRIMTYPE> | <INT TYPE> | ;                            */
/* <PRIMTYPE> => <PRIMITIVE> = <IDSTRING>                        */
/* <INT TYPE> => INTERNALS = <IDSTRING>                          */
/* '{' parsing is covered in this routine                        */

 TYP()
 {
  parseid(3) ;                        /* TYPES parsing           */
  while (1)                           /* <T> expansion           */
    {
     getid(rp,41) ;           /* 41 = missing { error    */
     find_token() ;
     if (toknn == 4)          /* if '{' found, then quit */
       {
        skip = 1 ;
        break ;               /* no types declared       */
       }
     if (toknn == 8)          /* <INT. TYPE> expansion   */
       IDSTRING(-2)  ;        /* -2 = code for internals */
     else      /* TYPES expansion    */
       {
        findprim() ;          /* <PRIMTYPE> expansion    */
        if (primid >= prim_count)
          error(30) ;         /* undefined function      */
        IDSTRING(primid) ;
       }     /* end TYPES   */
    }        /* end while 1 */
 }           /* end function*/
/*----------------------END TYP-----------------------------------*/
```

```
/*****************************************************************
 *                                                               *
 *                    IDSTRING SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* <IDSTRING> => id ; | <IDSTRING> id ,                          */

 IDSTRING(code)          /* code = 0 for inputs,-1 for outputs    */
 {                       /* -2 for internals.                     */
    int i ;
    while (delimiter != 2)          /* delimiter = ;       */
     {
      parseid(maxkey) ;             /* name                */
      update(code) ;                /* update symbol table */
                                    /* code = 0 for input  */
                                    /* -1 for output       */
                                    /* prim # for TYPE     */
      if (code == 0 )
          {
             act_val = act_val + 1 ;
             code_input(desc_no) ;     /* input code gen.     */
             i = sym_count - 1 ;
             symt[i].despos = 11 ;
             symt[i].delpos = 0 ;
          }
      if (code <= 0 )
        desc_no = desc_no + 1 ;
     }
   }
/*--------------------END IDSTRING-------------------------------*/

/*****************************************************************
 *                                                               *
 *                      CKT   SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/*----------------INTERCONNECTIONS parsing-----------------------*/
/* <CKT> => <IDSERIES> = <PRIM> ( <IDSERIES> ) ; <CKT> |         */
/*                        <IDSERIES> = <PRIM> ( <IDSERIES> ); */
/* <IDSERIES> => id | id, <IDSERIES>                             */

 CKT()
 {
  int outpar, end , i ;       /* number of output parameters          */
  int savpar, j , savinp ;
  int savid[maxouts], savn[maxouts], fwdp ;
  /* savid[] saves symbol table indexes while savn[] saves desc. */
  /* numbers for output list names                               */
  end = 0 ;
  while (end == 0)
   {
/*-----------------------<IDSERIES> for outputs-----------------*/
    outpar = -1 ;
    while (1)
     {
      getid(rp,46) ;      /* left side of assignment statement   */
      find_token() ;
      strcpy(inptab[inum].iname, token_buf);
      if (toknn == 5)     /*  if } found, end compilation        */
       {
        end = 1 ;
        break ;
       }
      if (toknn < maxkey) /* left hand side should not be a keyw.*/
        error(25) ;
      outpar = outpar + 1 ;
      findprim() ;
      if (primid < prim_count)
        error(25) ;       /* output name is a keyword            */
      findid() ;          /* find the symbol table index         */
```

```
      i = symid ;
      j = act_val ;
      if (i != j)          /* sort the SYMT table                    */
         {
            strcpy(temporary.name,symt[i].name);
            temporary.descno = symt[i].descno ;
            temporary.funcno = symt[i].funcno ;
            temporary.fanld = symt[i].fanld ;
            temporary.despos = symt[i].despos ;
            temporary.delpos = symt[i].delpos ;
            strcpy(symt[i].name,symt[j].name);
            symt[i].descno = symt[j].descno ;
            symt[i].funcno = symt[j].funcno ;
            symt[i].fanld = symt[j].fanld ;
            symt[i].despos = symt[j].despos ;
            symt[i].delpos = symt[j].delpos ;
            strcpy(symt[j].name,temporary.name);
            symt[j].descno = temporary.descno ;
            symt[j].funcno = temporary.funcno ;
            symt[j].fanld = temporary.fanld ;
            symt[j].despos = temporary.despos ;
            symt[j].delpos = temporary.delpos ;
            symid = j ;
         }
      act_val = act_val + 1 ;
      if (symid >= 0)
       {                       /* save output indices in an array */
        savid[outpar] = symid ;
        savn[outpar] = symt[symid].descno ;
       }
      valact = savid[0] ;
      if (delimiter == 4) /* '=' should follow the output names  */
        break ;
      else
       {
        if (delimiter != 1)
          error(31) ;          /* ',' expected after each name        */
       }
     }                         /* end while <IDSERIES> for outputs    */
/*-------------------------------------------------------------------*/
    if (end == 1)    /* if '}' found quit */
      break ;
/*-------------------------<PRIM>------------------------------------*/
    getid(rp,33) ;             /*  function name                     */
    if (delimiter != 6)        /*  '(' should follow function name   */
      error(29) ;
    if (err_count == 0)
     {
      if (outpar > 0)          /* if # of outputs > 1, connect exten- */
       {                       /* sion pointers.                      */
        for (j = 0; j < outpar; j = j + 1)
         {
          fprintf(tc, " 1 ]d 15 ]d", savn[j], savn[j+1]) ;
          fprintf(tc, " 1 ]d 16 ]d", savn[j+1], savn[0]) ;
          symt[valact].despos = symt[valact].despos + 8 ;
          fadvance(tc,8) ;
         }
       }
     }
    findprim() ;
    if (primid >= prim_count)
     {
      findid() ;               /* function name is a type */
      primid = symt[symid].funcno ;
     }
    inptab[inum].inp_num = primt[primid].numpar ;
    if (err_count == 0)
       {
          fprintf(tc," 33 ]d ]d",savn[0],primid) ;
          symt[valact].despos = symt[valact].despos + 3 ;
```

```
    }
  fadvance(3) ;
  if (primt[primid].outp != (outpar+1))
    error(35) ;          /* # of outputs should be as in table  */
  for(j = 0; j <= outpar; j = j + 1)   /* update symbol table  */
   {                                   /* and desc table       */
    symt[(savid[j])].funcno = primid ;
    updesct(token_buf, savid[j]) ;
   }
/*-------------------------------------------------------------*/
/*--------------------<IDSERIES> for inputs--------------------*/
  fwdp = 0 ;
  savpar = primt[primid].numpar ;  /* number of inputs         */
  savinp = 0 ;
  savprim = primid ;
  strcpy(savbuf, token_buf) ;  /* save function name/type      */
  while (savpar != 0)    /* while all inputs have been scanned */
   {
    parseid(maxkey) ;    /* parameter of function            */
    switch(savinp)
      {
          case 0 : strcpy(inptab[inum].inp1,token_buf);
                   break;
          case 1 : strcpy(inptab[inum].inp2,token_buf);
                   break;
          case 2 : strcpy(inptab[inum].inp3,token_buf);
                   break;
          case 3 : strcpy(inptab[inum].inp4,token_buf);
                   break;
          case 4 : strcpy(inptab[inum].inp5,token_buf);
                   break;
          case 5 : strcpy(inptab[inum].inp6,token_buf);
                   break;
          case 6 : strcpy(inptab[inum].inp7,token_buf);
                   break;
          case 7 : strcpy(inptab[inum].inp8,token_buf);
                   break;
          case 8 : strcpy(inptab[inum].inp9,token_buf);
                   break;
          case 9 : strcpy(inptab[inum].inp10,token_buf);
                   break;
      }
    savinp = savinp + 1 ;
    findid() ;           /* find parameter's location in the   */
    connect(0,savn,fwdp);/* generate code and update fanld     */
    if (savpar == 1)
     {
      if (delimiter != 5)  /* ')' expected after the last arg. */
       error(32) ;
     }
    savpar = savpar - 1 ;
    if (savpar != 0)
     {
      if (delimiter != 1)  /*, expected after first parameter  */
        error(31) ;
      parseid(maxkey) ;    /*  get next argument               */
      switch(savinp)
        {
          case 0 : strcpy(inptab[inum].inp1,token_buf);
                   break;
          case 1 : strcpy(inptab[inum].inp2,token_buf);
                   break;
          case 2 : strcpy(inptab[inum].inp3,token_buf);
                   break;
          case 3 : strcpy(inptab[inum].inp4,token_buf);
                   break;
          case 4 : strcpy(inptab[inum].inp5,token_buf);
                   break;
          case 5 : strcpy(inptab[inum].inp6,token_buf);
                   break;
```

```
            case 6 : strcpy(inptab[inum].inp7,token_buf);
                     break;
            case 7 : strcpy(inptab[inum].inp8,token_buf);
                     break;
            case 8 : strcpy(inptab[inum].inp9,token_buf);
                     break;
            case 9 : strcpy(inptab[inum].inp10,token_buf);
                     break;
          }
        savinp = savinp + 1 ;
        if (savpar > 1)
          {
           if (delimiter != 1)/* , expected after argument        */
             error(31) ;
          }
        else
          {
           if (delimiter != 5)
             error(32) ;       /* ) expected after last arg.      */
          }
        findid() ;             /* find symbol table index for the */
                               /* input name                      */
        connect(1,savn,fwdp); /* generate code and update fanld  */
        savpar = savpar - 1 ;
        fwdp = fwdp + 1 ;

        if (outpar > 0)              /* multioutput case */
          outpar = outpar - 1 ;
        else
          {
           if (savpar != 0)          /* multiinput case  */
             {
              fprintf(tc," 1 ]d 15 ]d",savn[fwdp - 1], desc_no) ;
              fprintf(tc," 1 ]d 16 ]d",desc_no, savn[fwdp-1]) ;
              symt[valact].despos = symt[valact].despos + 8 ;
              fadvance(tc,8) ;
              savn[fwdp] = desc_no ;
              desc_no = desc_no + 1 ;
             }
          }
       } /* end if                          */
     }   /* end of <IDSERIES> for inputs */
    inum = inum + 1;
  }      /* end while end = 0            */
}        /* end <CKT>                    */
/*--------------------END CKT-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                    CONNECT SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* Circular list generation for the circuit. Previous list is    */
/* broken and new circular loop is made.                         */

connect(f,savn,fwdp)
int f ;     /* f is 0 or 1 */
int savn[], fwdp ;      /* savn has desc # of output names */
{
 int i ;
 if (err_count == 0)
  {
   /*-----------------------update fanld-------------------------*/
   for (i = 0; i < normcount; i = i + 1)  /* find name in nort  */
    if (strcmp(nort[i].nom,savbuf) == 0)
      break;
   if (i < normcount)      /* if over ride is used for norm load */
     symt[symid].fanld = symt[symid].fanld + nort[i].nmld ;
   else                    /* use default value from prim. lib   */
     symt[symid].fanld=symt[symid].fanld + primt[savprim].normld;
```

```
    /*-------------------------------------------------------------*/
    fprintf(tc," 2 ]d", symt[symid].descno) ;
    fprintf(tc," 3 ]d", symt[symid].descno) ;
                        /* save current pointer from the input   */
    fprintf(tc," 1 ]d 8 ]d", symt[symid].descno, savn[fwdp]) ;
    fprintf(tc," 1 ]d 11 ]d", symt[symid].descno, f);
    fprintf(tc," 1 ]d 9 ]d", savn[fwdp], f) ;
    fprintf(tc," 1 ]d 12 ]d", savn[fwdp], f) ;
                              /* complete the C-list             */
    fprintf(tc," \n");
    symt[valact].despos = symt[valact].despos + 20 ;
    filecount = 0 ;
   }
 }     /* end connect  */
/*---------------------END CONNECT-------------------------------*/

/*****************************************************************
 *                                                               *
 *                    DEFINE  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* <DEF> => <PRIMID> : <DEFINITIONS>                             */
/* <DEFINITIONS> => RISEDELAY(num, num) = num  |                 */
/*                  FALLDELAY(num, num) = num  |                 */
/*                  FANOUT = num               |                 */
/*                  NORMLOAD = num             |                 */
/*                  OVERLOAD = num             |                 */
/*                  TECHNOLOGY = <TECHTYPE>                      */
/* <TECH TYPE> = TTL | NMOS | CMOS | ECL                         */

DEF()
 {
  int j, num, savtoken ;
  int fan1, tech1, over1 ;
  skip = 0 ;
  parseid(9) ;      /* DEFINE expected */
  while (1)
   {
    getid(rp,42) ;
    find_token();             /* function name or type         */
    if (toknn == 6)           /*   if token = 'INITIALIZE'     */
     {
      skip = 1;
      break ;
     }
    del_tab[index1].val = 0 ;
    strcpy(savbuf, token_buf);/*  save function name in savbuf  */
    findprim() ;
    if ( primid >= prim_count)
     {
      findid () ;
      primid = symt[symid].funcno ;
     }
    fan1  = primt[primid].fanout ;
    tech1 = primt[primid].technology ; /* default parameters    */
    over1 = primt[primid].overld ;

/*-----------------------DEFINE statement------------------------*/
    while (delimiter != 2)    /*  each DEFINE ends with ';'     */
     {
      getid(rp,43) ;
      find_token();
      savtoken = toknn ;      /*  savtoken = 'RISEDELAY' ...etc..*/
      switch(savtoken) {
        case 10: rfdel(0) ;               /* rise delay        */
                 break ;
        case 11: rfdel(1) ;               /* fall delay        */
                 break ;
        case 12: getid(rp,33) ;           /* TECHNOLOGY = --   */
                 for (j = 13; j <= 16; j = j + 1)
```

```
                {
                 if (strcmp(token_buf, keyword[j]) == 0)
                   break ;
                }
               if (j > 16)
                 error(36) ;            /* undefined technol.*/
               else
                 tech1 = j ;
               break ;
      case 17: getid(rp,33);/* FANOUT   -  get value of para.*/
               fan1 = atoi(token_buf) ; /* ASCII to integer  */
               break ;
      case 18: break ;     /* normload handled in first pass */
      case 19: getid(rp,33) ;
               over1 = atoi(token_buf) ;/* value of overld   */
               break ;
      default: error(36) ;  /* syntax error message */
    }     /* end switch  */
  }       /* while ;  each DEFINE ends with ';' */

  /*--------------------generate code for mode-----------------*/
  if (err_count == 0)
   {
    lim = 0;
    while (lim < dptr)
     {
      finddesc(savbuf) ;
      if (lim > dptr)
        break ;
      num = symt[descid].fanld ;
      if (num > fan1)
        {
           del_tab[index1].indx = index;
           del_tab[index1].typ_num = 12 ;
           del_tab[index1].num_ipt = 0;
           del_tab[index1].num_opt = 0;
           cmode(num, fan1, over1, tech1) ;
           index = index + 1 ;
           index1 = index1 + 1;
        }
     }
   }
  /*------------------------------------------------------------*/
 }        /* end while (1) */
}         /* end DEFINE    */
/*--------------------END DEFINE-------------------------------*/

/*****************************************************************
 *                                                               *
 *                     CMODE  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* Code generator for mode. Code is generated only if mode != 0  */

cmode(num, fan1, over1, tech1)
int num, fan1, over1, tech1 ;
{
 if (num <= (fan1 + over1))
  {
   if (tech1 == 16)
     fprintf(td," 1 ]d 6 2", symt[descid].descno) ;
   else
     fprintf(td," 1 ]d 6 1", symt[descid].descno) ;
  }
 else
  {
   if (tech1 == 16)
     fprintf(td," 1 ]d 6 3", symt[descid].descno) ;
   else
     fprintf(td," 1 ]d 6 4", symt[descid].descno) ;
```

```
    fadvance(td,4) ;
   }
  del_tab[index1].dsc_nb = symt[descid].descno ;
  del_tab[index1].val = del_tab[index1].val + 4 ;
 }   7* end CMODE */
/*--------------------END CMODE---------------------------------*/

/*****************************************************************
 *                                                               *
 *                    RFDEL  SUBROUTINE                          *
 *                                                               *
 *****************************************************************/
/* code generator for the delay modifications                    */

 rfdel(n1)
 int n1 ;
 {
  int par1, par2, parm1, parm2, num, savpar1, savpar2, inp ;
  if (delimiter != 6)
    error(29) ;          /* '(' expected after RD, FD */
  getid(rp,33) ;
  par1 = atoi(token_buf) ;             /* first index */
  parm1 = primt[primid].numpar ;
  parm2 = primt[primid].outp ;
  if (par1 > parm1)
    error(39) ;              /* incorrect first index  */
  if (delimiter != 1)        /* ',' expected           */
    error(31) ;
  getid(rp,33) ;
  par2 = atoi(token_buf) ;           /* second index */
  if (par2 > parm2)
    error(40) ;
  if (delimiter != 5)
    error(32) ;          /* ')' expected          */
  getid(rp,33) ;         /* value of rise delay  */
  num = atoi(token_buf) ;
  savpar1 = par1 ;         /* save first index     */
  savpar2 = par2 ;         /* save second index    */
  lim = 0 ;
  while ( lim < dptr)      /* generate code for all*/
   {                       /* descs. using name in */
    finddesc(savbuf) ;     /* savbuf               */
    del_tab[index1].dsc_nb = symt[descid].descno ;
    del_tab[index1].num_ipt = savpar1;
    del_tab[index1].num_opt = savpar2;
    lim = lim + parm2 - 1 ; /* lim is incremented by # of outputs*/
    if (lim > dptr)          /* desctable has successive entries  */
      break ;                /* for multi-output descriptors      */
    fadvance(tm,2) ;
    if (par1 > 1)          /* first access desc.   */
     {
      fprintf(tm," 6 ]d",symt[descid].descno);
      del_tab[index1].val = del_tab[index1].val + 2 ;
      del_tab[index1].indx = index;
      if (n1 == 0)
            del_tab[index1].typ_num = 10 ;
      else
            del_tab[index1].typ_num = 11 ;
      par1 = par1 - 2 ;
      while( par1 > 0)
       {
        fprintf(tm," 7 ");
        del_tab[index1].val = del_tab[index1].val + 1 ;
        fadvance(tm,1) ;
        par1 = par1 - 2 ;
       }
     }    /* if par1 > 1 */
    else                   /* if par1 = 1          */
     {
        fprintf(tm," 4 ]d",symt[descid].descno);
```

```
        del_tab[index1].val = del_tab[index1].val + 2 ;
        del_tab[index1].indx = index;
        if (n1 == 0)
                del_tab[index1].typ_num = 10 ;
        else
                del_tab[index1].typ_num = 11 ;
     }
    inp = savpar1 ] 2 ;   /* even or odd par1     */
    if (par2 == 0)
      {
         fprintf(tm," 5 ]d ]d", ((2+n1) + 2*inp), num) ;
         del_tab[index1].val = del_tab[index1].val + 3 ;
      }
    else                  /* multi-output case    */
     {
      par2 = par2 - 1 ;
      fprintf(tm," 8 ") ;
      del_tab[index1].val = del_tab[index1].val + 1 ;
      del_tab[index1].indx = index;
      if (n1 == 0)
            del_tab[index1].typ_num = 10 ;
      else
            del_tab[index1].typ_num = 11 ;
      fadvance(tm,3) ;
      while (par2 > 0)
       {
        fprintf(tm," 9 ") ;
        del_tab[index1].val = del_tab[index1].val + 1 ;
        fadvance(tm,1) ;
        par2 = par2 - 1 ;
       }
      if (inp  == 0)
       {
        if (n1 == 0)
          {
              fprintf(tm," 10 ]d",num) ;
              del_tab[index1].val = del_tab[index1].val + 2 ;
          }
        else
          {
              fprintf(tm," 11 ]d",num) ;
              del_tab[index1].val = del_tab[index1].val + 2 ;
          }
       }
      else
       {
        if (n1 == 0)
          {
              fprintf(tm," 12 ]d",num) ;
              del_tab[index1].val = del_tab[index1].val + 2 ;
          }
        else
          {
              fprintf(tm," 13 ]d",num) ;
              del_tab[index1].val = del_tab[index1].val + 2 ;
          }
        fadvance(tm,2) ;
       }
     }
    par1 = savpar1 ;
    par2 = savpar2 ;
    index = index + 1 ;
    index1 = index1 + 1 ;
   }
}       /* end RFDEL */
/*----------------------END RFDEL------------------------------*/
```

```
/*****************************************************************
 *                                                               *
 *                      MATGEN SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* Also generates default mode values for all functions involved */

 matgen()
 {
  int par1, par2, i, j, k, l, m, n ;
  int num, f1, o1, t1 ;
  char s[8];

  /*----------------------DEFAULT MODE GENERATION----------------*/
  for (i = 0; i < sym_count; i = i + 1)
   {
    if (symt[i].descno >= 0)
     {
      if (symt[i].funcno > 0)
       {
        descid = i ;      /* cmode() needs descid for code gen.   */
        num = symt[i].fanld ;
        f1 = primt[(symt[i].funcno)].fanout ;
        o1 = primt[(symt[i].funcno)].overld ;
        t1 = primt[(symt[i].funcno)].technology ;
        if (num > f1)     /* if non zero mode */
         {
            cmode(num, f1, o1, t1) ;
            symt[i].delpos = symt[i].delpos + 4 ;
         }
       }
     }
   }
  /*-------------------------------------------------------------*/
  /*-------------------DEFAULT DELAYS AND MATRIX STRUCTURE-------*/
  i = 0 ;
  while( i < dptr)
   {
    j = symt[desct[i].dnum].descno ;      /* descriptor number */
    k = symt[desct[i].dnum].funcno ;      /* function number   */
    n = desct[i].dnum ;
    strcpy(s, primt[k].nam2) ;    /* s contains name of function */
    bdread(s) ;       /* read block delays for s in rd/fdmat     */
    if (k >= 0)       /* if not input or types etc.*/
     {
      par1 = primt[k].numpar ;
      par2 = primt[k].outp   ;
      i =  i + par2 - 1 ;
           /* par1 = number of inputs, par2 = number of outputs  */
      for (l = 1; l <= par1; l = l + 2)   /* vertical scanning   */
       {
        if (l >  2)
         {
          if (l > 4)
            {
                fprintf(td," 7 ");
                symt[n].delpos = symt[n].delpos + 1 ;
            }
          else
            {
                fprintf(td," 6 ]d",j) ;
                symt[n].delpos = symt[n].delpos + 2 ;
            }
         }
        else            /* inputs = 1 or 2 */
            {
               fprintf(td," 4 ]d",j) ;
               symt[n].delpos = symt[n].delpos + 2 ;
            }
        fadvance(td,2) ;
```

```
    /*-----------------code for block delays-----------------*/
    if (rdmat[l-1][0] != -1)
      {
        fprintf(td," 5 2 ]d",rdmat[l-1][0]) ;
        symt[n].delpos = symt[n].delpos + 3 ;
      }
    if (fdmat[l-1][0] != -1)
      {
        fprintf(td," 5 3 ]d",fdmat[l-1][0]) ;
        symt[n].delpos = symt[n].delpos + 3 ;
      }
    if ((l+1) <= par1)
     {
      if (rdmat[l][0] != -1)
        {
          fprintf(td," 5 4 ]d",rdmat[l][0]) ;
          symt[n].delpos = symt[n].delpos + 3 ;
        }
      if (fdmat[l][0] != -1)
        {
          fprintf(td," 5 5 ]d",fdmat[l][0]) ;
          symt[n].delpos = symt[n].delpos + 3 ;
        }
     }
    fadvance(td,12) ;
    /*-------------------------------------------------------*/
    for (m = 2; m <= par2 ; m = m + 1)
     {
      if (m == 2)
        {
         fprintf(td," 14 ]d",matcount) ;
         symt[n].delpos = symt[n].delpos + 2 ;
         matcount = matcount + 1 ;
        }
      else       /* if number of outputs > 2 */
        {
         fprintf(td," 15 ]d ]d", matcount - 1, matcount) ;
         symt[n].delpos = symt[n].delpos + 3 ;
         matcount = matcount + 1 ;
        }
      fadvance(td,3) ;
      /*-----------------code for block delays---------------*/
      if (rdmat[l-1][m-1] != -1)
        {
          fprintf(td," 16 ]d ]d",matcount-1, rdmat[l-1][m-1]);
          symt[n].delpos = symt[n].delpos + 3 ;
        }
      if (fdmat[l-1][m-1] != -1)
        {
          fprintf(td," 17 ]d ]d",matcount-1,fdmat[l-1][m-1]) ;
          symt[n].delpos = symt[n].delpos + 3 ;
        }
      if ((l+1) <= par1)
        {
         if (rdmat[l][m-1] != -1)
           {
             fprintf(td," 18 ]d ]d",matcount-1,rdmat[l][m-1]) ;
             symt[n].delpos = symt[n].delpos + 3 ;
           }
         if (fdmat[l][m-1] != -1)
           {
             fprintf(td," 19 ]d ]d",matcount-1,fdmat[l][m-1]) ;
             symt[n].delpos = symt[n].delpos + 3 ;
           }
        }
      fadvance(td,12) ;
      /*-------------------------------------------------------*/
     }   /* end for m = -- */
  }      /* end for l = -- */
}        /* end if k >= 0  */
```

```
    i = i + 1 ;
   }          /* end for i = -- */
 }  /* end matgen */
/*--------------------END MATGEN-------------------------------*/

/*****************************************************************
 *                                                               *
 *                      BDREAD SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* This routine reads the block delays for a given function name */
 bdread(s)
 char s[8] ;
 {
  int i, j, num1, x, y, w, z, i1 ;
  FILE *r1 ;
  char p[8] ;
  r1 = fopen("d:bldel","r") ;      /* block delay file */
/*-----------------initialize delay matrix to -1 --------------*/
  for (i = 0; i < maxouts; i = i + 1)
   {
    for (j = 0; j < maxouts; j = j + 1)
     {
      rdmat[i][j] = -1 ;
      fdmat[i][j] = -1 ;
     }
   }
/*------------------read default block delays-----------------*/
  fscanf(r1,"]s",p) ;
  while (strcmp(p,"END") != 0)
   {
    fscanf(r1,"]d",&num1) ;
    for (i1 = 1; i1 <= num1; i1 = i1 + 1)
     {
      fscanf(r1,"]d ]d ]d ]d",&x, &y, &w, &z) ;
      if (strcmp(p,s) == 0)
       {
        rdmat[x][y] = w ;
        fdmat[x][y] = z ;
       }
     }
    if (strcmp(p,s) == 0)
      break ;
    fscanf(r1,"]s",p) ;
   }
  fclose(r1) ;
 }
/*--------------------END BDREAD-------------------------------*/

/*****************************************************************
 *                                                               *
 *                        INI  SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* <INI> => INITIALIZE : <INITSTRING>                            */
/* <INITSTRING> => id = num , <INITSTRING> | id = num ;          */

 INI()
 {
  int flag ;
  flag = 0 ;                    /* begint[0] = NULL indicator     */
  if (skip != 1)
   parseid(6) ;
  while (delimiter != 2)        /*  INITIALIZE ends with ';'      */
   {
    parseid(maxkey) ;
    if (delimiter != 4)         /* '=' expected after the name    */
      error(27) ;
    findid() ;                  /* symbol table index             */
    symt[symid].ini_num = ord_ini ;
```

```
    ord_ini = ord_ini + 1 ;
    getid(rp,43) ;              /*  initialization value          */
    if (err_count == 0)
     {
      fprintf(ti," 20") ;
      if (flag == 0)
        fprintf(ti," 21") ;
      else
        fprintf(ti," 22") ;
      /* allocate storage for tstack */
      fprintf(ti," 23 24 25") ;
      /* generate code */
      fprintf(ti," 26 ]d", symt[symid].descno) ;
      fprintf(ti," 27 ]s",token_buf) ;
      flag = 1 ;
      fadvance(ti,6) ;
     }
   }
 }
/*--------------------END INI-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                       PRI  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* <PRI> => PRINTOUT : <IDSTRING>                                */

PRI()
{
 int i, j ;
 parseid(7) ;
 i = 0 ;
 while (delimiter != 2)       /* PRINTOUT ends with ';'          */
  {
   parseid(maxkey) ;
   findid() ;
   symt[symid].pri_num = ord_pri ;
   if (err_count == 0)
    {
     for (j = 0; j <= 7 ; j = j + 1 )
      {
       if (token_buf[j] == '\0')
         break ;
       else
        {
         fprintf(tp," 28 ]d ]d ]c",i, j,token_buf[j]);
         val_prt = val_prt + 1 ;
         fadvance(tp,4) ;
        }
      }
     symt[symid].pri_val = val_prt ;
     ord_pri = ord_pri + 1 ;
     val_prt = 0 ;
     fprintf(tp," 29 ]d ]d",i,symt[symid].descno) ;
     fadvance(tp,3) ;
     i = i + 1 ;
    }
  }
 fprintf(tp," 30 31 ]d 32",i) ;
 fadvance(tp,4) ;
}
/*------------------END PRI-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                     STRCPY SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* copies one string in another                                  */
```

```
strcpy(s,t)             /* copies s = t  */
char *s, *t ;
 {
  while(*s++ = *t++)
   ;
 }
/*--------------------END STRCPY---------------------------------*/

/*****************************************************************
 *                                                               *
 *                     STRCMP SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* returns zero if string s is equal to string t    .            */

strcmp(s,t)
char s[], t[] ;
 {
  int i ;

  i = 0 ;
  while(s[i] == t[i])
    if (s[i++] == '\0')
      return (0) ;
  return(s[i] - t[i]) ;
 }
/*--------------------END STRCMP---------------------------------*/

/*****************************************************************
 *                                                               *
 *                      GETID SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* finds the next id in the user file                            */

getid(rx, ernm) /* finds the next id and returns it in token_buf */
int ernm ;      /* error number in case of EOF */
FILE *rx ;
 {
  int i , c, flag ;
  flag = 0 ;
  delimiter = -1 ;
  i = 0 ;
  while((delimiter < 1 ) || (flag == 0))
   {                        /* flag = 1 when some non blank char is */
                            /* read into token buffer               */
    c = fgetc(rx) ;
    buff[bb] = c ;          /* buff is the 80 character buffer for  */
    bb = bb + 1 ;           /* printing the entire line after it is */
    if (bb > 78)            /* read.     bb = index for buff        */
     {
      bb = 0 ;
     }
    switch(c)
     {
      case ' ' : delimiter = -1 ;
                 break ;
      case ',' : delimiter = 1 ;
                 break ;
      case ';' : delimiter = 2 ;
                 break ;
      case ':' : delimiter = 3 ;
                 break ;
      case '=' : delimiter = 4 ;
                 break ;
      case ')' : delimiter = 5 ;
                 break ;
      case '(' : delimiter = 6 ;
                 break ;
```

```
      case '\n': if (flag == 1)          /* flag = 1 indicates that */
                   delimiter = 7 ;       /* some non blank character*/
                                         /* was read into token_buf */
                 buff[bb-1] = '\0' ;

                                            /* output errors in the */
                 outerror() ;               /* line if any.         */
                 bb = 0 ;                   /* initialize line buff */
                 break ;
      case EOF : printf("]-s\n",buff) ;
                 error(ernm) ;
                 error(33) ;             /* EOF error               */
                 outerror() ;
                 exit(0) ;               /* abort the program       */
                 break ;

      default  : flag = 1 ;
                 delimiter = 0 ;
     }
    if (delimiter == 0 )
     {
      if (i <= 6 )
       {
        token_buf[i] = c ;
        i = i + 1 ;
       }
     }
   }
  token_buf[i] = '\0' ;
 }
/*----------------------END GETID---------------------------------*/

/*****************************************************************
 *                                                               *
 *                FIND_TOKEN SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* Token = maxkey if a nonkeyword name is encountered else it is */
/* equal to the index of the keyword in the keyword table        */

find_token()
 {
  int i ;
  for (i = 0; i < maxkey; i = i + 1)       /* sequential search  */
    if (strcmp (token_buf,keyword[i]) == 0 )
      break ;
  toknn = i ;        /* token = maxkey,   if match is not found  */
 }
/*-----------------END FIND_TOKEN---------------------------------*/

/*****************************************************************
 *                                                               *
 *                  PARSEID SUBROUTINE                           *
 *                                                               *
 *****************************************************************/
/* find the next identifier number                               */

parseid(i)
 int i ;                  /* i = token number to be compared to  */
  {
   getid (rp,33) ;              /* find the next identifier */
   find_token() ;               /* find token number        */
   if (toknn == maxkey)         /* check if name != function*/
    {
     findprim() ;
     if (primid < prim_count)
      error(25) ;               /* keyword found            */
    }
   if (toknn != i)              /* identifier of type 'i'   */
    error(i) ;                  /* expected.                */
```

```
  }
/*--------------------END PARSEID-------------------------------*/

/*****************************************************************
 *                                                               *
 *                     FINDID SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* finds the symbol table index. An error message is generated if*/
/* the name does not exist */

findid()
 {
  int i ;
  for (i = 0; i < sym_count; i = i + 1)
    if ( strcmp(token_buf,symt[i].name) == 0)
      break ;
  if (i == sym_count)       /* name not found in the symbol table */
    {
     error(28) ;            /* undeclared name                    */
     symid = -1 ;
    }
  else
    symid = i ;
 }
/*--------------------END FINDID--------------------------------*/

/*****************************************************************
 *                                                               *
 *                    FINDDESC SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* This routine is used in conjunction with the DEFINE parsing to*/
/* update all descriptors using the name in sbuf with the decl-  */
/* ared parameters.                                              */

 finddesc(sbuf)
 char sbuf[8] ;
  {
   int i ;
   for (i = lim; i < dptr; i = i + 1)
     if (strcmp(sbuf,desct[i].fun) == 0)
       break ;
    lim = i + 1 ;
    descid = desct[i].dnum ;
   }
/*--------------------END FINDDESC------------------------------*/

/*****************************************************************
 *                                                               *
 *                     UPDESCT SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* This routine updates the descriptor table.                    */
/* s = function name (type), num = symbol table index            */

 updesct(s, num)
 char s[8] ;
 int num ;
 {
  strcpy(desct[dptr].fun, s) ;
  desct[dptr].dnum = num ;
  dptr = dptr + 1 ;
 }
/*----------------------END UPDESCT-------------------------------*/
/*****************************************************************
 *                                                               *
 *                     FINDPRIM SUBROUTINE                       *
 *                                                               *
 *****************************************************************/
```

```
/* finds the primitive index                                          */

findprim()
 {
  int i ;
  for (i = 0; i < prim_count; i = i + 1)
    if ( strcmp(token_buf,primt[i].nam2) == 0)
      break ;
  primid = i ;
 }
/*----------------------END FINDPRIM-----------------------------*/

/*********************************************************************
 *                                                                   *
 *                    UPDATE  SUBROUTINE                             *
 *                                                                   *
 *********************************************************************/
/* This routine updates the symbol table. sym_count = symbol         */
/* table index, desc_no = descriptor number,                         */
/* typ = function type, 0 = input, -1 = output, -2 = internal        */

update(typ)
int typ ;
 {
   symt[sym_count].descno = desc_no ;
   symt[sym_count].funcno = typ ;
   strcpy(symt[sym_count].name, token_buf) ;
   del_tab[sym_count].dsc_nb = desc_no ;
   sym_count = sym_count + 1 ;
 }
/*----------------------END UPDATE-------------------------------*/

/*********************************************************************
 *                                                                   *
 *                CODE_INPUT SUBROUTINE                              *
 *                                                                   *
 *********************************************************************/
/* put the code for the input in the DCF file                         */

code_input(i)
int i ;                    /* i = desc_no */
 {
 int j;
  if (err_count == 0)
   {
    fprintf(tc," 33 ]d 0",i) ;
    j=hashf(token_buf);
    fprintf(tc," 1 ]d 0 ]d",i, j) ;
            /* parameters 0 and 1 contain the input line name */
    fprintf(tc," 1 ]d 2 1",i) ;
    fadvance(tc,15) ;
   }
 }
/*------------------END CODE_INPUT-------------------------------*/

/*********************************************************************
 *                                                                   *
 *                ERRMESSAGE SUBROUTINE                              *
 *                                                                   *
 *********************************************************************/
/* generates the error message                                        */

errmessage(i)
int i ;                 /* i = error number */
 {             /* err_ptr = global indicating error table index */
  printf("                                  ERROR   ") ;
  switch(i)
   {
    case 0 : printf(" 'MODULE' expected, ]s found\n",
                    errt[err_ptr].nm) ;
```

```
            break ;
case 1 : printf(" 'INPUTS' expected, ]s found\n",
                errt[err_ptr].nm) ;
            break ;
case 2 : printf(" 'OUTPUTS' expected, ]s found\n",
                errt[err_ptr].nm) ;
            break ;
case 3 : printf(" 'TYPES' expected, ]s found\n",
                errt[err_ptr].nm) ;
            break ;
case 4 : printf(" '{' expected, ]s found\n",
                errt[err_ptr].nm) ;
            break ;
case 5 : printf(" '}' expected, ]s found\n",
                errt[err_ptr].nm) ;
            break ;
case 6 : printf(" 'INITIALIZE' expected,]s found\n",
                errt[err_ptr].nm) ;
            break ;
case 7 : printf(" 'PRINTOUT' expected, ]s found\n",
                errt[err_ptr].nm) ;
            break ;
case 9 : printf(" 'DEFINE' expected, found ,]s \n"
                ,errt[err_ptr].nm) ;
            break ;
case 25: printf(" name ]s is a keyword\n",
                errt[err_ptr].nm) ;
            break ;
case 26: printf(" count = ]d, >>COMPILATION discontinued\n",
                                                err_count);
            break ;
case 27: printf(" '=' expected after ]s\n",
                errt[err_ptr].nm) ;
            break ;
case 28: printf(" ]s is undeclared\n",
                errt[err_ptr].nm) ;
            break ;
case 29: printf(" '(' expected after ]s\n",
                errt[err_ptr].nm) ;
            break ;
case 30: printf(" ]s is undefined function\n",
                errt[err_ptr].nm) ;
            break ;
case 31: printf(" ',' expected after ]s\n",
                errt[err_ptr].nm) ;
             break ;
case 32: printf(" ')' expected after ]s\n",
                errt[err_ptr].nm) ;
             break ;
case 33: printf(" unexpected EOF\n");
             break ;
case 34: printf(" missing END\n") ;
            break ;
case 35: printf(" incorrect # of args. on LHS ]s\n"
                , errt[err_ptr].nm) ;
            break ;
case 36: printf(" in syntax, ]s unrecognized \n",
                errt[err_ptr].nm) ;
            break;
case 37: printf(" count = 10, >>Compilation discontinued\n");
            break ;
case 38: printf(" = 0,  ***END OF COMPILATION***\n") ;
            break ;
case 39: printf(" 1st DELAY index is > lim\n");
            break ;
case 40: printf(" 2nd DELAY index is > lim\n");
            break ;
case 41: printf(" missing {\n") ;
            break ;
case 42: printf(" missing INITIALIZE\n") ;
```

```
            break ;
    case 43: printf(" missing ';'\n") ;
            break ;
    case 44: printf(" undefined function\n") ;
            break ;
    case 45: printf(" missing TYPES\n") ;
            break;
    case 46: printf(" missing }\n") ;
            break ;
    case 47: printf(" missing ')'\n");
            break;
    case 48: printf(" missing DEFINE\n");
            break ;
    case 49: printf(" incorrect # of input arguments in call\n");
            break ;
    case 50: printf(" incorrect # of out arguments in call\n");
            break ;
   }
  err_count = err_count + 1 ;
  if (err_count > 9)
   {
    error(37) ;
    outerror() ;
    exit(0) ;
   }
 }
/*------------------END ERRMESSAGE-------------------------------*/

/*****************************************************************
 *                                                               *
 *                 OUTERROR  SUBROUTINE                          *
 *                                                               *
 *****************************************************************/
/* This routine outputs all errors encountered in a line after   */
/* entire line has been read.                                    */

 outerror()
  {
   int i;
   i = 0 ;
   while (err_ptr >= 0)       /* if error count for a line is > 0 */
    {                         /* print all errors encountered     */
     errmessage(errt[i].errno) ;
     i = i + 1 ;
     err_ptr = err_ptr - 1 ;
    }
  }
/*--------------------END OUTERROR-------------------------------*/

/*****************************************************************
 *                                                               *
 *                     ERROR  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* This routine enters the error number and the name of the wrong*/
/* identifier in the errt (error table). The errors are printed  */
/* after the whole line has been scanned.                        */

 error(i)
  int i ;       /* i = error number */
  {
   err_ptr = err_ptr + 1 ;          /* error count for one line  */
   errt[err_ptr].errno = i ;        /* errno = error number      */
   strcpy(errt[err_ptr].nm,token_buf) ;
                                    /* copy name of wrong identi.*/
  }
/*--------------------END ERROR----------------------------------*/
```

```
/*****************************************************************
 *                                                               *
 *                    FIRSTP SUBROUTINE                          *
 *                                                               *
 *****************************************************************/
/* This routine scans the user program for any expansion requests*/
/* if found, they are put on a expt stack along with their func #*/
/* Note that func no. may not be known at this time              */

 firstp()
 {
 extern int maxpid;
 int i,j,option,tpid,found;
 char templ[8], target[8];
 r1=fopen("d:pi1","r");
  search(r1, r1, -1, 9, 0,48) ; /* search for DEFINE (9)     */
                                /* 48 = missing DEFINE error */
  getid(r1,42) ;       /* 42 = missing INITIALIZE error */
  find_token() ;
  while (toknn!=6)        /* search for EXPAND */
     {
     if (toknn == 21)     /* EXPAND */
        {
        sfound=0;
        getid(r1,33) ;         /* this is what we expand;  33 = EOF error */
        strcpy(expt[expcount].fname, token_buf) ;
        strcpy(templ,token_buf);       /* save name for later */
        expcount++;
        getid(r1,33);        /* how far do we expand# */
        findprim();
        if (primid<prim_count)    /* a valid primitive# */
           {
           strcpy(target,primt[primid].nam2);    /* yes, save it as is */
           tpid=primid;
           }
        else
           {
           found=0;                                /* no, what type is it# */
           for (i=0; i<=maxpid; i++)
              {
              j=0;
              while (typelist[i][j].used==1)
                 {
                 if (strcmp(token_buf,typelist[i][j].sname)==0)  /* match# */
                    {
                    strcpy(target,primt[i].nam2);   /* primitive name */
                    tpid=1;                         /* primitive id */
                    found=1;
                    break;
                    }
                 else
                    j++;
                 }   /* end while */
              if (found==1)
                 break;
              }      /* end for */
           if ((i>maxpid) && (found==0))
              error(44);
           }         /* end else */
        check_deeper(templ,tpid,target);
        if (sfound==0)
           {
           printf("I couldn't find ]s in any of the circuit descriptions.\n",
                                                 target);
           printf("\n\nDo you wish to: \n\n");
           printf("   1.  Continue anyway\n");
           printf("   2.  Give up\n\n");
           printf("Enter the number of your selection ");
           scanf("]d",&option);
           if (option==2)
```

```
            exit(0);
          }
        }  /* end if toknn=21 */
      getid(r1,42);                  /* look for INITIALIZE */
      find_token();
      }                              /* end while toknn */
 multi_mod();                          /* do multimodule case after  */
 fclose(r1);                           /* enumerated EXPANDs         */
 }
/*----------------------END FIRSTP-------------------------------*/

/*****************************************************************
 *                                                               *
 *                    SECONDP SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* second pass routine, expansion is carried out in this routine */
/* First the specs for user program are copied to P1EXP. The func*/
/* # for function to be expanded is determined at the same time. */
/* Next, the function's description is copied to SCR1. The user  */
/* program is searched for any call to function to be expanded,  */
/* The code from SCR1 is substituted at this point. The expanded */
/* circuit is stored in SCR2. Finally P1EXP, SCR2 are appended   */
/* along with the simulation specs from user program.            */

 secondp(expnum)
 int expnum ;        /* expnum = expand table index */
 {
  int i, j ;
  count = 0 ;
/*--search for TYPES in user prog (copy to P1EXP)----------------*/
  search( r1, w1, -1, 3, 1,45) ; /* 45 = missing TYPES error     */

/*----------search if function to be expanded is a TYPE----------*/
  while (1)           /* search function to be expanded. Replace it*/
   {                  /* with x's. Find its fnum and put in expt   */
    getid(r1,41) ;    /* 41 = missing { error */
    find_token() ;
    if (toknn == 4 )        /* '{' then break */
      break ;
    pdelim(w1) ;            /* print to P1EXP */
    if (toknn != 8)         /* function name, not an INTERNAL */
     {
      findprim() ;
      if (primid >= prim_count)
       error(30) ;
      while (1)             /* search function's name in TYPE list */
       {
        getid(r1,43) ;      /* 43 = missing ; error */
        if (strcmp(token_buf, expt[expnum].fname) == 0)
         {
          expt[expnum].fnum = primid ;
          strcpy(token_buf,"XXXXXXX") ;
         }
        pdelim(w1) ;
        if (delimiter == 2)     /* if ';' then end of TYPE list   */
          break ;
       }                    /* end type search */
     }
    else    /* INTERNALS */
     {
      search(r1, w1, 2, -1, 1,43);/* print internals as is */
                                  /* 2 = delim ';'  (error 43)*/
     }
   }       /* end TYPES */
/*-----------------find primid for the function-----------------*/
  if (expt[expnum].fnum == -1)/* if no types,then find function #*/
   {
    strcpy(token_buf,expt[expnum].fname) ;
    findprim() ;
```

```
    if (primid >= prim_count)
      error(30) ;
    else
      expt[expnum].fnum = primid ;
   }
/*-------------------------------------------------------------*/
  fclose(r1) ;
/*---------------------find function in lib---------------------*/
  r2 = fopen("d:lib1","r") ;    /* library */
  s1 = fopen("d:scr1","w") ;    /* scratch file for function */
  while (1)       /* find the function to be expanded in the lib */
   {
    getid(r2,44); /* 44 = function not found in lib */
    find_token() ;
    if (toknn == 0)    /* MODULE */
     {
      getid(r2,33) ;
      if (strcmp(token_buf, primt[(expt[expnum].fnum)].nam2) == 0)
        {
        break ;
        }
     }
   }
/*-----------------function found-------------------------------*/
/*--------store inputs, outputs on  stacks----------------------*/
  getid(r2,33) ;          /* INPUTS */
  incount = 0 ;
  while (1)
   {
    getid(r2,43) ;        /* 43 = missing ; error */
    find_token() ;
    if (toknn == 2)       /* OUTPUTS */
      break ;
    else
     {
      strcpy(instack[incount],token_buf) ;
      incount = incount + 1 ;
     }
   }
  outcount = 0 ;
  while (1)    /* OUTPUTS */
   {
    getid(r2,45) ;   /* 45 = missing TYPES error */
    find_token() ;
    if (toknn == 3)  /* TYPES */
      break ;
    else
     {
      strcpy(outstack[outcount],token_buf) ;
      outcount = outcount + 1 ;
     }
   }
/*---------------TYPES should be tacked by function name---------*/
/*---------------save types on a typstack-----------------------*/
  typcount = 0 ;
  while (1)
   {
    getid(r2,41) ;        /* 41 = missing { error */
    find_token() ;
    if (toknn == 4)       /* {         */
      break ;
    else
     {
      if (toknn == 8)     /* INTERNALS */
       {
        pdelim(s1) ;            /* internals are copied on to SCR1*/
        search(r2,s1,2,-1,1,43);/* so that they can be reproduced*/
                                /* 'occurance' times at the end   */
       }
      else /* TYPE declarations */
```

```
      {
       pdelim(w1) ;    /* function types are tacked by the name  */
       while (1)       /* of the function to be expanded         */
        {              /* They are saved on a stack so that it   */
                       /* can be determined in its desc. if a    */
                       /* type is used (will be tacked similarly)*/
         getid(r2,43) ;  /* 43 = missing ; error */
         strcpy(typstack[typcount], token_buf) ;
         typcount = typcount + 1 ;
         for (i = 0; i < 5; i = i + 1)
          {
           if (token_buf[i] == '\0')
             break ;
          }
         for (j = i; j < 5; j = j + 1)
           token_buf[j] = ' ' ;
         token_buf[5] = primt[expt[expnum].fnum].nam2[0] ;
         token_buf[6] = primt[expt[expnum].fnum].nam2[1] ;
         token_buf[7] = '\0' ;
         pdelim(w1) ;
         if (delimiter == 2)      /* ; */
           break ;
        }
      }
    }
  }
  fprintf(s1," { \n") ;
  count = 0 ;
/*-----------------------------------------------------------------*/
/*-------------write structural desc. to SCR1--------------------*/
  while (1)
   {
    search(r2, s1, 4, 5, 1,46);/* 4 = '='  , 5 = '}'       */
              /* write structural description to scr file */
              /* Types are tacked by function's name */
    if (toknn== 5)
      break ;
    getid(r2,46) ;/* 46 = missing } error    */
    ftype() ;
    if (ft != 0)  /* indicates type declared */
     {
      for (i = 0; i < 5; i = i + 1)
       {
        if (token_buf[i] == '\0')
         break ;
       }
      for (j = i; j < 5; j = j + 1)
        token_buf[j] = ' ' ;
      token_buf[5] =  primt[expt[expnum].fnum].nam2[0] ;
      token_buf[6] =  primt[expt[expnum].fnum].nam2[1] ;
      token_buf[7] = '\0' ;
     }
    pdelim(s1) ;
    search(r2, s1, 5, -1, 1,47);  /* 5 = ')' */
                                  /* outputs */
    count = 0 ;
   }
  fclose(w1) ;
  fclose(s1) ;
  fclose(r2) ;
  count = 0 ;
  cexpand(expnum);/*pand the primitive whenever it occurs in ckt */
 }
/*--------------------END SECONDP--------------------------------*/

/*****************************************************************
 *                                                               *
 *                     PDELIM SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
```

```
/* prints delimiter on the file                                    */

 pdelim(wx)
 FILE *wx ;
 {
  fprintf(wx," ]s",token_buf) ;
  count = count + 1 ;
  switch(delimiter)
   {
    case 8 : fprintf(wx,"  ") ;
             break ;
    case 7 : fprintf(wx," \n") ;
             count = 0 ;
             break ;
    case 6 : fprintf(wx," (") ;
             break ;
    case 5 : if (print_select==0)
                {
                fprintf(wx," )");
                break;
                }
             else
                {
                fprintf(wx," ) ; \n");
                count=0;
                break;
                }
    case 4 : fprintf(wx," =") ;
             break ;
    case 3 : fprintf(wx," :") ;
             break ;
    case 2 : fprintf(wx," ;\n") ;
             count = 0 ;
             break ;
    case 1 : fprintf(wx," ,") ;
             break ;
   }
  if (count >= 7)
   {
    fprintf(wx," \n") ;
    count = 0 ;
   }
 }
/*----------------------END PDELIM-------------------------------*/

/*****************************************************************
 *                                                               *
 *                      FCOPY SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* copies one file in another                                    */

fcopy(rr, ww)
FILE *rr, *ww ;
{
 int c ;
 while ((c = getc(rr)) != EOF)
   putc(c,ww) ;
}
/*----------------------END FCOPY--------------------------------*/

/*****************************************************************
 *                                                               *
 *               COPY_NOEND  SUBROUTINE                          *
 *                                                               *
 *****************************************************************/
/* copies one file in another without the EOF                    */

copy_noend(inf,outf)
FILE *inf,*outf;
```

```
{
int pdone;
   pdone=0;
   print_select=1;
   getid(inf);                          /* read the token from file */
   find_token();                        /* see what it is           */
   while (pdone==0)                     /* quit when we see END     */
      {
      switch (toknn)
         {
         case 3:   pdelim(outf);        /* write the TYPES token */
                   getid(inf,33);       /* and get the next one  */
                   find_token();
                   if (toknn==4)        /* check for no TYPES    */
                      {
                      fprintf(outf," ; \n");  /* no types, start new line */
                      }
                   break;
         case 6:   print_select=1;
                   pdelim(outf);         /* write the INITIALIZE token */
                   getid(inf,33);        /* check for sequence of */
                   find_token();         /* INITIAL:; PRINTOUT:   */
                   if (toknn==7)         /* is this PRINTOUT#     */
                      fprintf(outf," ; \n");  /* yes, start new line */
                   break;
         case 9:   print_select=0;
                   pdelim(outf);         /* write the DEFINE token */
                   getid(inf,33);        /* check for sequence of */
                   find_token();         /* DEFINE:; INITIAL:; PRINTOUT: */
                   if (toknn==6)         /* is this INITIAL# */
                      {
                      fprintf(outf," ; \n");  /* yes, print new line */
                      pdelim(outf);           /* then print INITIAL  */
                      getid(inf,33);
                      find_token();
                      if (toknn==7)           /* is this PRINTOUT# */
                         fprintf(outf," ; \n"); /* yes, print new line */
                      }
                   break;
         case 20:  pdone=1;
                   break;
         default:  pdelim(outf);          /* write the token */
                   getid(inf,33);         /* and get the next one */
                   find_token();
                   break;
         }    /* end switch (toknn) */
      }    /* end while (toknn) */
   print_select=0;
}
/*--------------------END COPY_NOEND-----------------------------*/

/*****************************************************************
 *                                                               *
 *                    REVERSE SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* reverses a string                                             */

reverse (s)
char s[] ;
  {
   int c, i, j ;
   for (i = 0, j = 6; i < j; i++, j--)
     {
      c = s[i] ;
      s[i] = s[j] ;
      s[j] = c ;
     }
  }
/*-------------------END REVERSE-------------------------------*/
```

```
/*****************************************************************
 *                                                               *
 *                        ITOA  SUBROUTINE                       *
 *                                                               *
 *****************************************************************/
/* changes a integer value to ASCII code                         */

itoa(n, s)
char s[] ;
int n ;
{
 int i ;
 i = 0 ;
 do {
     s[i++] = n ] 10 + '0' ;
    } while ((n /= 10) > 0) ;
   reverse(s) ;
}
/*----------------------END ITOA-------------------------------*/

/*****************************************************************
 *                                                               *
 *                       FTYPE SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* It finds if a type for the function name in the primitive's   */
/* description has been declared. ft = 1 indicates this          */

 ftype()
  {
   int i ;
   ft = 0 ;
   for (i = 0; i < typcount; i = i + 1)
    {
     if (strcmp(token_buf, typstack[i]) == 0)
      {
       ft = 1 ;
       break ;
      }
    }
  }
/*---------------------END FTYPE-------------------------------*/

/*****************************************************************
 *                                                               *
 *                     CEXPAND SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* expands a primitive for user's request                        */

 cexpand(exnum) /* expand primitive's occurance */
 int exnum ;    /* exnum = expand table index   */
 {
  int i ;
  s2 = fopen("d:scr2","w") ;    /* expanded desc */
  r1 = fopen("d:p11", "r") ;    /* user prog     */
  occurance = 0 ; /* # of times call to function is made */
                  /* internals will be duplicated this # */
  search(r1,r1,-1,4,0,41);   /* find circuit description */
                             /* 4 =  { */
/*-------------------------------------------------------------*/
  skip = 0 ;
  while (1)
   {
    ckt_line(&out1count, r1,out1stack, &in1count, in1stack, &skip) ;
    if (found_end==1)              /* find an ENDSWAP while in cktline#*/
      {
       found_end=0;                /* sure did, set up for the next one*/
       fprintf(s2," ENDSWAP ; \n");  /* write the keyword onto SCR2  */
```

```
        }
    if (found_start==1)            /* find a SWAPLIN while in cktline# */
        {
        found_start=0;             /* yes, get ready for the next one */
        fprintf(s2," SWAPLIN ;\n"); /* write the keyword onto SCR2     */
        }
    if (skip == 1)
      break ;
/*-------------------------------------------------------------------*/
/*--------------------print same, or substitute code-----------------*/
    if (strcmp(savfunc, expt[exnum].fname) == 0)
     {
      if (out1count != outcount)
        error(50) ;
      if (in1count != incount)
        error(49) ;
      if (err_count == 0)
        {
         substitute() ;    /* substitute the primitive's code */
         occurance = occurance + 1 ;
        }
     }
    else                /* print the code as is              */
     {
      for (i = 0; i < out1count ; i = i + 1)
        {
         strcpy(token_buf, out1stack[i]) ;
         delimiter = 1 ;   /*   */
         if (i == (out1count-1))
           delimiter = 4 ; /* =    */
         pdelim(s2) ;      /* SCR2 */
        }
      strcpy(token_buf, savfunc) ;
      delimiter = 6 ;    /* ( */
      pdelim(s2) ;
      for (i = 0; i < in1count; i = i + 1)
        {
         strcpy(token_buf, in1stack[i]) ;
         delimiter = 1 ;
         if (i == (in1count-1))
           delimiter = 5 ; /* ) */
         pdelim(s2) ;
        }
      count = 0 ;
      fprintf(s2," ;\n") ;
     }   /* end else */
   }
  fclose(s2) ;
 }
/*--------------------END CEXPAND--------------------------------*/

/*********************************************************************
 *                                                                   *
 *                SUBSTITUTE SUBROUTINE                              *
 *                                                                   *
 *********************************************************************/
/* copies function's code from s1 to s2. Internals are tacked by */
/* occurance number */

 substitute()
 {
  int i, skip1 ;
  s1 = fopen("d:scr1", "r") ;   /* function's description */
  search(s1,s1,-1, 4, 0,41) ;      /* find description */
                                   /* 4 = '{'          */
 /*-----------------------OUTPUTS-----------------------------*/
  skip1 = 0 ;
  while (1)
   {
    ckt_line(&out2count, s1, out2stack, &in2count, in2stack,&skip1) ;
```

```
    if (skip1 == 1)
      break ;
/*------------------write to SCR2 (s2)-------------------------*/
/*inputs/outputs are replaced by corresponding names from in/out1*/
    for (i = 0; i < out2count; i = i + 1)
     {
      foutorder(out2stack[i]) ;
      if (outorder != -1)
        {
        strcpy(token_buf, out1stack[outorder]) ;
        }
      else
       {
        finorder(out2stack[i]) ;
        if (inorder != -1 )
          {
          strcpy(token_buf, in1stack[inorder]) ;
          }
        else
          tack(occurance,out2stack[i]) ;
       }
      delimiter = 1 ;
      if (i == (out2count - 1))
        delimiter = 4 ;        /* = */
      pdelim(s2) ;
     }
    strcpy(token_buf, savfunc) ;
    delimiter = 6 ;    /* ( */
    pdelim(s2) ;
    for (i = 0; i < in2count; i = i + 1)
     {
      finorder(in2stack[i]) ;
      if (inorder != -1)
        {
        strcpy(token_buf, in1stack[inorder]) ;
        }
      else
       {
        foutorder(in2stack[i]) ;
        if (outorder != -1)
          {
          strcpy(token_buf, out1stack[outorder]) ;
          }
        else
          tack(occurance,in2stack[i]) ;
       }
      delimiter = 1 ;
      if (i == (in2count - 1))
        delimiter = 5 ;   /* )  */
      pdelim(s2) ;
     }
    fprintf(s2," ;\n") ;
    count = 0 ;
   }   /* end while */
  fclose(s1) ;
 }
/*------------------END SUBSTITUTE-----------------------------*/

/*****************************************************************
 *                                                               *
 *                  FOUTORDER SUBROUTINE                         *
 *                                                               *
 *****************************************************************/

 foutorder(s)
 char s[] ;
  {
   int i ;
   for (i = 0; i < outcount; i = i + 1)
    {
```

```
     if (strcmp(s, outstack[i]) == 0)
       break ;
     }
    if (i >= outcount)
     outorder = -1 ;
    else
     outorder = i ;
   }
/*------------------END FOUTORDER-------------------------------*/

/*****************************************************************
 *                                                               *
 *                   FINORDER SUBROUTINE                         *
 *                                                               *
 *****************************************************************/

 finorder(s)
 char s[] ;
  {
   int i ;
   for (i = 0; i < incount; i = i + 1)
    {
     if (strcmp(s, instack[i]) == 0)
       break ;
     }
    if (i >= incount)
     inorder = -1 ;
    else
     inorder = i ;
   }
/*------------------END FINORDER--------------------------------*/

/*****************************************************************
 *                                                               *
 *                   CKT_LINE SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* This routine reads one line of circuit description and stores */
/* inputs, outputs in proper arrays                              */

ckt_line(oocount, rx, oostack, iicount, iistack, skip1)
int *oocount, *iicount, *skip1 ;
FILE *rx ;
char oostack[][8], iistack[][8] ;
 {
                   /* Outputs */
  *oocount = 0 ;
  while (1)      /* put outputs on out stack */
    {
     getid(rx,46);/* 46 = missing } error     */
     find_token() ;
     if (toknn==24)                          /* did we find an ENDSWAP#*/
        {
        found_end=1;                         /* yes, tell CEXPAND      */
        getid(rx,46);                        /* get the next token     */
        find_token();                        /* and see what it is     */
        }
     if (toknn==23)                          /* find a SWAPLIN too#    */
        {
        found_start=1;                       /* yes, tell CEXPAND      */
        getid(rx,46);
        find_token();
        }
     if (toknn == 5)  /* } */
      {
       *skip1 = 1 ;
       break ;
      }
     strcpy(oostack[*oocount], token_buf) ;
     *oocount = *oocount + 1 ;
```

```
    if (delimiter == 4)   /* = */
      break ;
   }     /* end while outputs */
/*-------------------------------------------------------------*/
  if ((*skip1) != 1)
   {
    getid(rx,33) ;            /* function name */
    strcpy(savfunc, token_buf) ;
/*--------------------------INPUTS-----------------------------*/
    *iicount = 0 ;
    while(1)  /* inputs */
     {
      getid(rx,47) ;  /* 47 = missing ')' error */
      strcpy(iistack[*iicount], token_buf) ;
      *iicount = *iicount + 1 ;
      if (delimiter == 5)    /* ) */
        break ;
     }
   }
 }
/*--------------------END CKT_LINE-----------------------------*/

/*****************************************************************
 *                                                               *
 *                    SEARCH  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* searchs by a defined string                                   */

search(xi, xo, delm, tokm, fg,ernum)
FILE *xi, *xo ;
int delm, tokm, fg, ernum ;
{
 while (1)
  {
   getid(xi,ernum) ;
   if (fg == 1)
    {
    pdelim(xo) ;
    if (delimiter == 5)
     {
      fprintf(xo," ;\n") ;
      count = 0 ;
     }
    }
   if (delimiter == delm)
     break ;
   find_token() ;
   if (toknn == tokm)
     break ;
  }
}
/*---------------------END SEARCH------------------------------*/

/*****************************************************************
 *                                                               *
 *                      TACK  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/

tack(occ,iobuff)
int occ ;
char iobuff[8] ;
{
 int i, j ;
 strcpy(token_buf, iobuff) ;
 for (i = 0; i < 7; i = i + 1)
  {
   if (token_buf[i] == '\0')
     break ;
```

```
  }
 for (j = i; j < 7; j = j + 1)
  token_buf[j] = ' ' ;
 token_buf[7] = '\0' ;
 reverse(token_buf) ;
 itoa(occ, token_buf) ;
}
/*--------------------END TACK-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                  MULTI_MOD SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* This routine handles sub modules. All sub-modules are tacked  */
/* to the library (STRUC) in the front. Also expand table is     */
/* incremented for each sub-module.                              */

multi_mod()
{
 li1 = fopen("d:lib1","w") ;
 while(1)
  {
   getid(r1,34) ;    /* 34 = missing END error */
   find_token() ;
   if (toknn == 20)  /* END */
     break ;
   if (toknn == 0)   /* MODULE */
    {
     pdelim(li1) ;
     getid(r1,33) ;  /* sub module name */
     strcpy(primt[prim_count].nam2, token_buf) ;
     strcpy(expt[expcount].fname, token_buf) ;
     expt[expcount].fnum = prim_count ;
     prim_count = prim_count + 1 ;
     expcount = expcount + 1 ;
     pdelim(li1) ;
     search(r1, li1, -1, 5, 1,46); /* 5 = } */
    }
  }
t1 = fopen("d:struc","r") ;
 fcopy(t1,li1) ;
 fclose(t1) ;
 fclose(r1) ;
 fclose(li1) ;
}
/*------------------END MULTI_MOD-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                   FADVANCE SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* advances a number of spaces in the file and returns to the    */
/* beginning of the next line                                    */

 fadvance(t1,numm)
  int numm ;
  FILE *t1 ;
  {
   filecount = filecount+  numm ;
   if (filecount > 22)
    {
     filecount = 0 ;
     fprintf(t1," \n") ;
    }
  }
/*--------------------END FADVANCE-----------------------------------*/
```

```
/*****************************************************************
 *                                                               *
 *                      HASHF  SUBROUTINE                        *
 *                                                               *
 *****************************************************************/

hashf(s)    /* finds hash value for string s  */
char *s;
{
 int hashval ;
 for (hashval = 0; *s != '\0' ;)
    hashval += *s++ ;                /* convert from string to # */
 test(&hashval);                     /* and test for collision */
 hashtable[hashcount]=hashval;
 hashcount++;
 return (hashval) ;
}
/*---------------------END HASHF-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                       TEST  SUBROUTINE                        *
 *                                                               *
 *****************************************************************/

test(value)
int *value;
{
   int i;
   for (i=0; i<hashcount; i++)
      {
      if (hashtable[i]==(*value))          /* hashtable collision# */
         {
         (*value)=(*value)+11;             /* yes, add a prime number..*/
         test(value);                      /* and test again         */
         }
      }
}
/*----------------------END TEST-----------------------------------*/
```

APPENDIX B
THE PRECOMP ROUTINES

```
/*****************************************************************
 *                                                               *
 *                  Precompilation Routines                      *
 *                     for the compiler                          *
 *          including:  structural expansion handling            *
 *                      USING handling                           *
 *       to be used with the version 3.1 of the CADD program     *
 *                                                               *
 *****************************************************************/

#include " lc\stdio.h"      /*<stdio.h> in DOS 3.x environment   */

#define maxkey 29             /* maximum number of keywords      */
#define maxprim 100           /* maximum number of primitives    */
#define maxouts 32            /* maximum of 32 outputs per prim  */

/*------------------GLOBAL DECLARATIONS-------------------------*/
extern int strcpy() ;       /*   copies one string to another */
extern int strcmp() ;       /*   string comparison            */
extern int getid()  ;       /*   get next identifier          */
extern int find_token() ;   /*   returns the token number     */
                            /*   (keyword table index)        */
                            /*   the given function name/type */
extern int findprim() ;     /*   finds primitive library index*/
extern int pdelim() ;       /*   prints a file of keywords and */
                            /*   tokens, separated by delimiters*/
extern int fcopy() ;        /*   file copying routine         */
extern int copy_noend();    /*   copies everything but END token */
extern int ckt_line() ;     /*   scans one line of ckt desc.  */
extern int search() ;       /*   search a delimiter or toknn  */
extern int multi_mod();
extern int secondp();
extern int tack();
/*-------------------------------------------------------------*/

/*----------------DATA STRUCTURES--------------------------*/
struct prim_tab {           /*  Primitive table :          */
  char nam[8] ;             /*  primitive table            */
  char nam2[8] ;            /*  EXTENDED primitive table   */
  int numpar, outp ;        /*  numpar = no. of parameters */
  int normld, fanout ;      /*  for the function           */
  int technology, overld ; /*  outp = # of outputs        */
 } ;

struct namepair {           /* this holds system-generated  */
  char e_from[8];           /* expansion requests           */
  char e_to[8];
 };

struct swapname {       /* each of these nodes will contain a module */
  char sname[8];            /* specified in the USING parameter list */
  int used;                 /* this indicates whether used or not    */
  };

struct functable {          /* each of these nodes will contain      */
  char fnname[8];           /* a function and level for a library    */
  int level;                /* VOHL module                           */
  };

struct exp_tab {            /* expansion table  */
  char fname[8];
```

```
  int fnum;
  };
/*-----------------------------------------------------------*/


/*---------------STORAGE ALLOCATION-------------------------*/
extern struct prim_tab primt[maxprim], *primptr ;
extern struct swapname typelist[maxprim][8];  /*table of replacements*/
extern struct swapname swaplist[20][8];  /* USING parameter storage */
extern struct namepair reqtable[maxprim];
extern struct exp_tab expt[30];

extern char userprg[8];
extern int expcount;
extern int sfound;
extern int req_count;       /* number of system expand reqs */
extern int line_count;      /* line index for swaplist      */
int line_item;              /* index within single line     */
extern int swap_flag;       /* indicates whether all of this*/
                            /* is necessary or not          */
int expand_flag;            /* indicates need for additional*/
                            /* expansion declarations       */
int found_start;            /* indicates presence of SWAPLIN*/
int found_end;              /* indicates presence of ENDSWAP*/
extern int add_flag;        /* set if cell is to be added to*/
                            /* primitive library            */
extern int delimiter, bb, skip ;  /* delimiter = delimiter type   */
                            /* bb = buff[80] index (line)   */
extern int toknn ;      /* err_count  = error count     */
extern int cellcount;               /* # of MODULEs in user program */
extern int prinpflag;       /* controls printing of source  */
extern int print_select;    /* determines whether a ')' causes*/
                            /* a linefeed or not (default=no)*/
extern int prim_count, primid ;    /* prim_count = # of primitives */
extern int sys_prims;               /* number of system-defined prims*/
extern int features[maxprim][2];     /* used to describe the type of */
                            /* descriptions available;              */
                            /* -1 -> empty     0 -> block only    */
                            /* 1 -> d:struc only  2 ->block & struc */
extern int append_table[maxprim]; /* keeps track of the functions*/
extern int append_index;    /* we've added to the user program    */
int modules_to_do;          /* number of STRUC-only additions to do */
extern int expdone ;        /* marks completion of STRUC expansion*/
extern int no_compilation; /* used when only making library additions*/
extern char token_buf[8], savbuf[8], buff[80] ; /* buff = 1 line  */
extern char keyword[maxkey][8] ;  /* keyword table                */
extern char instack[maxouts][8], outstack[maxouts][8] ;
extern int incount , outcount ;
extern int count ;              /* for printing on the file */
extern char savfunc[8] ;
int maxpid;

extern FILE *r1 ;               /* pointer to input data file        */
extern FILE *r2 ;               /* pointer to STRUCT library         */
extern FILE *r3 ;
extern FILE *r4 ;
extern FILE *w1 ;               /* pointer to expanded file   P1EXP  */
extern FILE *t1 ;
extern FILE *s1 ;
extern FILE *s2 ;
/*-----------------------------------------------------------------*/

/*****************************************************************
 *                                                               *
 *                     COUNTCELLS subroutine                     *
 *                                                               *
 *****************************************************************/
countcells(lookfile)
FILE *lookfile;
{
```

```
   int start_looking;
   start_looking=0;
   getid(lookfile,33);
   find_token();
   if ((toknn>25) && (toknn<maxkey))      /* first keyword an add key?*/
      start_looking=1;                    /* yes, start looking for   */
                                          /* a missing DEFINE.        */
   while (toknn!=5)                       /* take a look at the 1st   */
      {                                   /* module...                */
      if (toknn==0)                       /* find a MODULE token? */
         cellcount++;                     /* yes, update count    */
      getid(lookfile,33);
      find_token();
      }
   getid(lookfile,33);                    /* read the next token      */
   find_token();                          /* should be a DEFINE       */
   if ((toknn!=9) && (start_looking==1)) /* is it?  if not...        */
      {
      no_compilation=1;                 /* this is a cell addition only.*/
      printf("no compilation this pass\n");
                                          /* note--if this is just a  */
      }                                   /* missing DEFINE, let Ausif*/
                                          /* handle it. After all, he */
                                          /* wrote this thing <grin>. */
   while (toknn!=20)                      /* scan the rest of the ckt */
      {
      if (toknn==0)                       /* find a MODULE token? */
         cellcount++;                     /* yes, update count    */
      getid(lookfile,33);
      find_token();
      }
}
/*---------------------END COUNTCELLS-----------------------------*/

/*****************************************************************
 *                                                               *
 *                          BUILD subroutine                     *
 *                                                               *
 *****************************************************************/
build()                                /* build a VOHL file with no */
   {                                   /* END keyword               */
   FILE *adds,*specs;                  /* modules added go here     */
   int adflag, current_primitive;      /* used locally for each module */
   int i,j,skip1,done;
   int found, divert;
   char buffer1[8];                    /* temporary string buffer   */
   add_flag=0;
   adflag=0;
   done=0;
   divert=0;
   maxpid=0;                           /* largest primid encountered */
   print_select=1;                     /* newline after ')'         */
   adds=fopen("d:newckts.vhl","w");
   specs=fopen("d:specs","w");
   getid(r1,33);
   find_token();
   while (done==0)
      {
      if ((toknn>25) && (toknn<maxkey))
         {
         add_flag=1;
         adflag=1;
         pdelim(adds);         /* save the keyword in addition file*/
         getid(r1,33);
         find_token();
         }
      switch(toknn)
         {
         case 0:   divert=0;          /* save MODULE keyword     */
                   pdelim(r2);
```

```
            if (adflag==1)
               {
               count--;
               pdelim(adds);   /* and to the new ckt file     */
               }
            getid(r1,33);
            find_token();
            break;
 case 3:    pdelim(r2);         /* save the TYPES keyword      */
            if (adflag==1)
               {
               count--;
               pdelim(adds);   /* and to the new ckt file     */
               }
            while (1)
               {
               getid(r1,33);
               find_token();
               if ((toknn==8) || (toknn==4))
                                  /* quit for INTERNA or { */
                  break;
               pdelim(r2);
               find_token();   /* find out what we just read  */
               findprim();     /* it might be a prim, so check */
               if (primid < prim_count) /*if this is a primitive*/
                  {
                  current_primitive=primid;
                                  /* next IDs will be of this type*/
                  if (primid>maxpid)
                     maxpid=primid;
                  }
               else
                  {            /* not a primitive, so add it */
                  i=0;
                  while(typelist[current_primitive][i].used==1)
                     i++;     /* look for next free space*/
                  strcpy(typelist[current_primitive][i].sname,
                                                   token_buf);
                  typelist[current_primitive][i].used=1;
                  }
               } /* while (1) */
            if (toknn==4)
               {
               count=0;              /* if a '{', then no TYPES */
               fprintf(r2," ; \n"); /* so we need to save a ';' */
               if (adflag==1);
                  fprintf(adds, " ; \n");
               pdelim(r2);            /* write the '{' token   */
               if (adflag==1)
                  pdelim(adds);
               getid(r1,33);
               find_token();
               }
            break;
 case 5: if (adflag==1)    /* if token is a '}' then */
               {                  /* that's the end of this */
              adflag=0;           /* module, so print '}' and */
              fprintf(adds," } \n"); /* clear the addition flag*/
               }
         pdelim(r2);
         getid(r1,33);
         find_token();
         break;
 case 6: uexpand();             /* print system expand reqs  */
         print_select=1;        /* put linefeeds after )'s  */
         divert=0;
         pdelim(r2);            /* print INITIAL             */
         getid(r1,33);
         find_token();
         if (toknn==7)
```

```
            fprintf(r2," ; \n ");
          break;
case 9:   print_select=0;      /* no linefeed after ')'     */
          pdelim(r2);          /* print DEFINE token        */
          getid(r1,33);        /* read the next token       */
          find_token();
          switch(toknn)        /* anything to DEFINE?      */
             {
             case 6: print_select=1;  /* no, INITIAL found  */
                     divert=0;
                     fprintf(r2," ; \n");/* start a new line */
                     uexpand();
                     pdelim(r2);       /* and print INITIAL */
                     getid(r1,33);
                     find_token();
                     if (toknn==7) /*  no INITIALized params */
                        fprintf(r2," ; \n");
                     break;
             case 21: divert=0;        /* user wants EXPANDs   */
                     fprintf(r2," ; \n"); /*start a new line*/
                      uexpand();       /* put ours first though*/
                      pdelim(r2);      /* now print user's     */
                      getid(r1,33);
                      find_token();
                      break;
             default: divert=1;        /* DEFINE params present */
                      fprintf(r2," ; \n");  /* start new line */
                      fprintf(specs,"DEFINE: ");
                      pdelim(specs);
                      getid(r1,33);
                      find_token();
                      break;
             }
          break;
case 20:  done=1;              /* END token, so quit       */
          break;
case 21:  divert=0;            /* EXPAND   */
          pdelim(r2);
          getid(r1,33);
          find_token();
          break;
case 22:  line_item=0;         /* USING-- here we go..     */
                               /* temporary file           */
          fprintf(r2,"SWAPLIN ; \n");  /* mark this ckt line*/
          swap_flag=1;                 /*we'll be doing swaps*/
          expand_flag=1;               /*and probably calling*/
          found=0;
          do                           /* for expansions     */
             {
             getid(r1,47);     /* get the parameter list */
             find_token();
             if (toknn==25)            /* is this a NOEXP?   */
                {
                expand_flag=0;      /* don't need to call for*/
                getid(r1,47);       /* expansions            */
                }
             else                     /* expand to this TYPE */
              {
              if (found==0)
                {
                for (i=0; i<=maxpid; i=i+1)
                   {
                   j=0;              /* need to find it in list*/
                   while (typelist[i][j].used==1)
                    {
                    if (strcmp(token_buf,
                        typelist[i][j].sname)==0)  /*a match?*/
                       {
                       strcpy(buffer1,primt[i].nam2);/*yes,save*/
                       found=1;
```

```
                        break;              /* and continue */
                        }
                     else
                        j++;  /* no, look at next TYPE */
                     }  /* end while typelist*/
                    if (found==1)
                      break;
                    }    /* end for  */
                  }    /* end if found */
                }      /* end else */
               strcpy(swaplist[line_count][line_item].sname,
                                    token_buf); /*one by one */
               swaplist[line_count][line_item].used=1;
               line_item++;
               } while (delimiter !=5);    /* stop for a ')' */
            ckt_line(&outcount,r1,outstack,&incount,instack,
                                                  &skip1);
            for (i=0; i<outcount-1; i++)      /* write outputs*/
               {
               fprintf(r2," ]s ,",outstack[i]);
               if (adflag==1)
                  fprintf(adds," ]s ,",outstack[i]);
               }
            fprintf(r2," ]s = ",outstack[i]);
            if (adflag==1)
               fprintf(adds," ]s = ",outstack[i]);
            fprintf(r2,"]s(",savfunc); /*write the function name*/
            if (adflag==1)
               fprintf(adds,"]s(",savfunc);
            for (i=0; i<incount-1; i++)   /* write the inputs*/
               {
               fprintf(r2," ]s ,",instack[i]);
               if (adflag==1)
                  fprintf(adds," ]s ,",instack[i]);
               }
            fprintf(r2," ]s ) ;\n",instack[i]);
            if (adflag==1)
               fprintf(adds," ]s ) ;\n",instack[i]);
            if (expand_flag==1)   /* need to call for expand? */
               {
               for (i=0; i<req_count; i++)     /* yes, see if */
                  {                            /* already did */
                  if (strcmp(reqtable[i].e_from,savfunc)==0)
                     break;                 /* if already marked this*/
                  }                         /* one then ignore it   */
               if (i==req_count)
                  {
                  strcpy(reqtable[i].e_to,buffer1);
                  strcpy(reqtable[i].e_from,savfunc);
                  req_count++;
                  expand_flag=0;
                  }
               }
            fprintf(r2," ENDSWAP ; \n");
            line_count++;                  /* increment swaplist */
            getid(r1,33);                /* then read next token*/
            find_token();
            break;
default:    if (divert==0)
               {
               pdelim(r2);
               if (adflag==1)
                  {
                  count--;
                  pdelim(adds);
                  }
               }
            else
               pdelim(specs);
```

```
                    getid(r1,33);       /* and read the next one    */
                    find_token();
                    break;
            }  /* switch */
        }  /*while*/
    fclose(r1);
    fprintf(adds," END ; \n");      /*      END the file         */
    fclose(adds);
    fprintf(specs," END ; \n");      /*      END the file         */
    fclose(specs);
    print_select=0;                 /* restore to initial state  */
    }
/*--------------------END BUILD---------------------------------*/

/*****************************************************************
 *                                                               *
 *                      UEXPAND  subroutine                      *
 *                                                               *
 *****************************************************************/
/* This routine prints any system-generated expansion requests  */
/* onto the file r2 (OUTFILE).                                  */
uexpand()
{
int i;
for (i=0; i<req_count; i++)
 fprintf(r2," EXPAND: ]s : ]s ;\n",reqtable[i].e_from,reqtable[i].e_to);
}
/*--------------------END UEXPAND-------------------------------*/

/*****************************************************************
 *                                                               *
 *                 STRUC_EXPAND  subroutine                      *
 *                                                               *
 *****************************************************************/
struc_expand()
{
int skip,passdown;
modules_to_do=cellcount;
while (expdone==0)
    {
    getid(r1,41);                   /* scan along until found { */
    find_token();
    while (toknn!=4)                  /* looking for { (indicates */
      {                               /* we're in the circuit proper) */
      getid(r1,41);                   /* scan along until found { */
      find_token();
      }
    skip=0;
    while (skip==0)               /* repeat until we find a '}' */
       {
       ckt_line(&outcount,r1,outstack,&incount,instack,&skip);
       if (skip==1)
          {
          break;                      /* quit        */
          }
       else
          {
          strcpy(token_buf,savfunc);    /* get the function name and*/
          findprim();                   /* see if it is a primitive  */
          if (primid<prim_count)        /* yes, but is it a valid one?*/
             {                          /* if not, let Ausif handle it*/
             if (features[primid][0]==1)   /* STRUC-only description? */
                {
                passdown=primid;
                append(passdown,r2);      /*add it to work file */
                }  /* if features */
             } /* if primid */
          }  /* else */
       }        /*while skip */
    modules_to_do--;                  /* hey, finished one         */
```

```
   cellcount--;                        /* decrement the cell count  */
   if (cellcount==0)                   /* cell count =0?            */
      {
      fprintf(r2, "END; \n");          /* yes, place the END on the */
      fclose(r1);                      /* expanded file and close it*/
      fclose(r2);
      if (modules_to_do==0)            /* modulesTodo also = 0?     */
         expdone=1;                    /* yes, then we're finished  */
      else
         {                             /* no, further expansion is  */
         r1=fopen("d:outfile","r");    /* required--make a new file */
         r2=fopen("d:infile","w");     /* the old OUTFILE becomes   */
         fcopy(r1,r2);                 /* the new INFILE            */
         fclose(r1);
         fclose(r2);
         r1=fopen("d:infile","r");     /* build the new OUTFILE by  */
         r2=fopen("d:outfile","w");    /* copying everything but the*/
         copy_noend(r1,r2);            /* END token                 */
         fclose(r1);
         r1=fopen("d:infile","r");
         countcells(r1);               /* count the # of cells in   */
         fclose(r1);                   /* this new input file       */
         r1=fopen("d:infile","r");     /* now that we've got the files*/
         struc_expand();               /* straight, expand recursively*/
         }
      }   /*  else  */
   }
}
/*--------------------END STRUC_EXPAND--------------------------*/

/*****************************************************************
 *                                                               *
 *                      APPEND  subroutine                       *
 *                                                               *
 *****************************************************************/
append(ptarget,writefile)
   int ptarget;
   FILE *writefile;
   {
   int aindex,done;
   FILE *lookfile;
   done=0;
   aindex=0;
   print_select=1;                      /* put linefeeds after ')' */
   do
      {
      if (append_table[aindex]==ptarget)   /* already added this one?*/
         break;                            /* yes, skip              */
      aindex++;
      } while (aindex<=append_index);
   if (aindex>append_index)                /* no, append this prim  */
      {
      lookfile=fopen("d:struc","r");
      do
         {
         getid(lookfile,33);            /* no, wrong primitive*/
         findprim();
         find_token();
         if (toknn==0)                          /* found a MODULE? */
            {                                /* yes, is it the right one?*/
            getid(lookfile,33);             /* get the primitive name*/
            findprim();
            if ((primid<prim_count) && (primid==ptarget))
               {
               done=1;                           /* yes, quit          */
               fprintf(writefile,"MODULE : ");
               pdelim(writefile);
               getid(lookfile,33);
               find_token();
               while (toknn!=3)
```

```
                {
                pdelim(writefile);
                getid(lookfile,33); /*write everything down to TYPES*/
                find_token();
                }
             pdelim(writefile);       /* write the TYPES token */
             getid(lookfile,33);
             find_token();
             if (toknn==4)            /* no TYPES ? */
                {
                fprintf(writefile, " ; \n");    /* start new line */
                fprintf(writefile, " { \n");    /* print the {    */
                getid(lookfile,33);             /* then grab token*/
                find_token();
                }
             while (toknn!=5)                /* copy down to '}'   */
                {
                pdelim(writefile);          /* copy the token*/
                getid(lookfile,33);       /* and get the next one */
                find_token();
                }
             pdelim(writefile);               /*write the '}'  */
             append_table[append_index]=primid; /*update the table */
             modules_to_do++;        /* I hate recursion          */
             append_index++;
             }
          }
       } while (done==0);
     fclose(lookfile);
     } /* if */
  print_select=0;                    /* restore to previous state */
  }    /* append */
/*--------------------END APPEND--------------------------------*/

/*****************************************************************
 *                                                               *
 *                        SWAP  subroutine                       *
 *                                                               *
 *****************************************************************/
/* This routine is invoked after expansion and just before*/
/* compilation.  If there are any requests for function   */
/* name substitution with USING, they will be handled     */
/* here.  The affected lines are delimited by SWAPLIN and */
/* ENDSWAP keywords which will be removed prior to compil-*/
/* ation.                                                 */
swap()
{
int skip1, found, i, j;
int perform, swaps;
r1=fopen("d:p11","r");             /* raw file is P11 */
w1=fopen("d:outfile","w");         /* processed file goes here */
getid(r1,41);
find_token();
while (toknn!=4)
   {
   pdelim(w1);                     /* copy down to the '{'  */
   getid(r1,41);
   find_token();
   }
pdelim(w1);                        /* write the  '{'        */
swaps = -1;
found_start=0;
found_end=0;
skip1=0;
while (1)
   {
   ckt_line(&outcount,r1,outstack,&incount,instack,&skip1);
   if (skip1==1)                   /* quit when we see a '}'  */
      break;
   if (found_end==1)       /* find an ENDSWAP while getting ckt line*/
```

```
      {
      perform=0;                    /* no need to check for swaps */
      found_end=0;                /* set up for the next ENDSWAP key */
      }
   if (found_start==1)              /* find a SWAPLIN keyword?      */
      {
      perform=1;                   /* need to start looking for swaps */
      found_start=0;               /* set up for next SWAPLIN key  */
      swaps++;                     /* select the next line of swaplist */
      }
   if (perform==1)              /* check for swaps on this ckt line? */
      {
      strcpy(token_buf,savfunc); /*move function name to token buffer*/
      findprim();                  /* and see what type it is        */
      i=0;                         /* initialize USING param selector */
      if (typelist[primid][0].used==1) /* this TYPE is user-defined?*/
         {
         found=0;                  /* yes, initialize to "not found"*/
         while (swaplist[swaps][i].used==1) /*match USING param list.*/
            {
            j=0;
            while (typelist[primid][j].used==1)  /*against TYPE list*/
               {
               if (strcmp(swaplist[swaps][i].sname,
                           typelist[primid][j].sname)==0)/*a match?*/
                  {
                  strcpy(savfunc,typelist[primid][j].sname);/*yes,swap*/
                  found=1;        /* tell outside world we're done */
                  break;                  /* and quit           */
                  }
               else
                  j++;                    /* no, look at next TYPE */
               }              /* while typelist*/
            if (found==1)                /* are we done? */
               break;          /* yes, look for next substitution  */
            else
               i++;                 /* no, try next USING parameter */
            }   /* while swaplist */
         if (found==0)
           {
           printf("couldn't find the module you wanted to replace ]s\n",
                                    savfunc);
           printf("compilation continues\n\n");
           }
         }  /* if typelist */
      }   /* if perform */
      for (i=0; i<outcount-1; i++)                    /* write outputs*/
         fprintf(w1," ]s ,",outstack[i]);
      fprintf(w1," ]s = ",outstack[i]);
      fprintf(w1,"]s(",savfunc);          /* write the function name */
      for (i=0; i<incount-1; i++)                  /* write the inputs*/
         fprintf(w1," ]s ,",instack[i]);
      fprintf(w1," ]s ) ;\n",instack[i]);
   }  /* while not skip  */
pdelim(w1);                              /* write the '}'          */
fcopy(r1,w1);                            /* then copy rest of file*/
fclose(r1);
fclose(w1);
r1=fopen("d:outfile","r");
r2=fopen("d:specs","r");
w1=fopen("d:p11","w");
print_select=1;
getid(r1,33);
find_token();
while (toknn!=5)
   {
   pdelim(w1);                 /* copy circuit body */
   getid(r1,33);
   find_token();
   }
```

```
pdelim(w1);
print_select=0;
getid(r2,33);
find_token();
if (toknn==20)
   fprintf(w1,"DEFINE: ;\n");
else
   {
   while (toknn!=20)
      {
      pdelim(w1);                /* copy the DEFINEd parameters */
      getid(r2,33);
      find_token();
      }
   }
while (toknn!=6)              /* look for the INITIAL:   */
   {
   getid(r1,33);
   find_token();
   }
pdelim(w1);
fclose(r2);
fcopy(r1,w1);                          /* copy the processed file */
fclose(r1);                            /* back to p11             */
fclose(w1);
}
/*--------------------END SWAP-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                  CHECK_DEEPER subroutine                      *
 *                                                               *
 *****************************************************************/
check_deeper(fnam,targetpid,targetname)
char fnam[];
int targetpid;
char targetname[];
{
FILE *lib;
int findex,skip;
int incnt,outcnt, i, intable;
char instk[maxouts][8], ostk[maxouts][8] ;
struct functable ftable[maxprim];
lib=fopen("d:struc","r");
getid(lib,33);
find_token();
while (1)
   {
   if (toknn==0)                  /* find a MODULE? */
      {
      getid(lib,41);              /* yes, is it the right one? */
      if (strcmp(fnam,token_buf)==0)
         {
         while (toknn!=4)         /* yes, read down to circuit body */
            {
            getid(lib,41);
            find_token();
            }
         break;
         }
      }
   getid(lib,44);
   find_token();
   }
findex=0;
skip=0;
while(1)
   {
   ckt_line(&outcnt,lib,ostk,&incnt,instk,&skip);
   if (skip==1)
```

```
      break;
   strcpy(ftable[findex].fnname,savfunc);
   strcpy(token_buf,savfunc);
   findprim();
   ftable[findex].level=features[primid][1];
   findex++;
   }
fclose(lib);
for (i=0; i<findex; i++)
   {
   checktable(i, &intable, ftable);    /* this one in expand table? */
   if ((ftable[i].level > features[targetpid][1]) && (intable==0))
      {
      strcpy(expt[expcount].fname,ftable[i].fnname); /*add it to table*/
      expcount++;
      check_deeper(ftable[i].fnname,targetpid,targetname);
      }
   else
      {
      if ((ftable[i].level==features[targetpid][1]) &&
                  (strcmp(ftable[i].fnname,targetname)==0))
         {          /* function name what we're looking for? */
         sfound=1;       /* yes, found at least one occurrance */
         }
      }
   }
}
/*--------------------END CHECK_DEEPER--------------------------*/

/*****************************************************************
 *                                                               *
 *                    CHECKTABLE subroutine                      *
 *                                                               *
 *****************************************************************/
checktable(index,result,table)
int index,*result;
struct functable table[];
{
int i;
*result=0;                    /* initialize to "not found"  */
for (i=0; i<expcount; i++)
   {
   if (strcmp(expt[i].fname,table[index].fnname)==0)
      {
      *result=1;
      break;
      }
   }
}
/*--------------------END CHECKTABLE----------------------------*/

/*****************************************************************
 *                                                               *
 *                  PERFORM_EXPANSION subroutine                 *
 *                                                               *
 *****************************************************************/
perform_expansion()
{
extern int occurance;
int i,k,edone,end;
for (k = 0; k < expcount; k = k + 1)  /* each expansion request*/
   {                                  /* handled one at a time */
   r1 = fopen("d:p11","r");
   w1 = fopen("d:p1exp","w") ;/* declaration part of expanded*/
                               /* user program                */
   secondp(k) ;                /* second pass - expansion     */
   fclose(r1) ;                /* expanded desc. is in SCR2   */
                               /* expanded declaration inP1EXP*/
   /*--------------copy P1EXP to temp-----------------------*/
   t1 = fopen("d:temp","w") ;
```

```
   r1 = fopen("d:p1exp","r") ;
   fcopy(r1,t1) ;
   fclose(r1) ;
   /*----------copy internals of function to t1 with tacking----*/
   s1 = fopen("d:scr1","r") ;   /* function's description    */
   while (1)    /* read internals and copy to d:temp with tacking */
      {
      getid(s1,41);        /* 41 = missing { error */
      find_token() ;
      if (toknn == 4)      /* { */
         break ;
      if (toknn != 8)      /* INTERNAL */
         {
         end = 0 ;
         for (i = 0; i < occurance; i = i + 1)
            {
            tack(i, token_buf) ;
            if (delimiter == 2)    /* ';'   */
               {
               end = 1 ;
               if ((i + 1) != occurance)
                  delimiter = 1 ;
               }
            pdelim(t1) ;
            if (end == 1)
            delimiter = 2 ;
            }
         }
      else
         pdelim(t1) ;
      }                       /* end while */
   fprintf(t1," {  \n") ;
   fclose(s1) ;
   /*------------append SCR2 to d:temp and copy back to P11--*/
   s2 = fopen("d:scr2","r") ;
   fcopy(s2,t1) ;
   fprintf(t1," }\n") ;
   fclose(s2) ;
   fclose(t1) ;
   t1 = fopen("d:temp","r") ;
   w1 = fopen("d:p11","w") ;
   fcopy(t1,w1) ;
   fclose(t1) ;
   fclose(w1) ;
   }  /* end for */
if (expcount == 0)
   printf(" >>> No expansion request \n");
else   /* tack last part (simulation control specs) to P11 */
   {
   r1=fopen("d:p11","r");
   t1 = fopen("d:temp","w") ;
   fcopy(r1,t1);
   fclose(r1);
   r1 = fopen(userprg,"r") ;/* user program   */
   getid(r1,33);
   find_token();
   while (toknn!=5)
      {
      getid(r1,33);
      find_token();
      }
   edone=0;
   getid(r1,33);                /* get next token after } */
   find_token();
   while (1)
      {
      switch(toknn)
         {
         case 0:  edone=1;          /* MODULE */
                  break;
```

```
            case 6:  print_select=1;
                     pdelim(t1);
                     getid(r1,34);
                     find_token();
                     if (toknn==7)
                        fprintf(t1," ; \n");
                     break;
            case 9:  print_select=0;
                     pdelim(t1);
                     getid(r1,34);
                     find_token();
                     if (toknn==6)
                        {
                        fprintf(t1," ; \n");
                        pdelim(t1);
                        getid(r1,34);
                        find_token();
                        if (toknn==7)
                           fprintf(t1," ; \n");
                        }
                     break;
            case 20: edone=1;          /* END */
                     break;
            case 21: getid(r1,34);     /* EXPAND */
                     getid(r1,34);
                     find_token();
                     break;
            default: pdelim(t1);
                     getid(r1,34);
                     find_token();
                     break;
            }   /* switch*/
         if (edone==1)
            break;
         }     /* while */
      fclose(t1) ;
      t1 = fopen("d:temp","r") ;
      w1 = fopen("d:p11","w") ;
      fcopy(t1,w1) ;
      fclose(t1) ;
      fclose(w1) ;
      }
   }
   /*-----------------END PERFORM_EXPANSION--------------------*/

/*****************************************************************
 *                                                               *
 *                      ADD_LIB subroutine                       *
 *                                                               *
 *****************************************************************/
add_lib()
{
int i,icnt,ocnt,done;
int skip,maxlevel;
done=0;
print_select=1;
r1=fopen("d:newckts.vhl","r");
getid(r1,33);                  /* find out which addition to do */
find_token();
do
   {
   icnt=0;
   ocnt=0;
   maxlevel=0;
   switch (toknn)
      {
      case 20: done=1;                 /* found the END token */
               break;
      case 26: printf("ADDCELL not installed yet....\n");
               break;
```

```
case 27: printf("Performing a structure-only addition for.. ");
         r2=fopen("d:temp","w");
         r3=fopen("d:struc","r");
         fcopy(r3,r2);
         fclose(r3);
         getid(r1,33);
         pdelim(r2);                    /* write MODULE token */
         getid(r1,33);                  /* get the module name */
         strcpy(primt[sys_prims].nam2,token_buf);
         printf("]s\n",token_buf);
         pdelim(r2);
         getid(r1,33);                  /* get INPUTS keyword */
         pdelim(r2);
         getid(r1,33);                  /* get first input name */
         find_token();
         while (toknn!=2)          /* stop when find OUTPUTS  */
            {
            icnt++;                  /* count and write inputs */
            pdelim(r2);
            getid(r1,33);
            find_token();
            }
         primt[sys_prims].numpar=icnt;
         pdelim(r2);                    /* write OUTPUTS keyword */
         getid(r1,33);
         find_token();                  /* get first output  */
         while (toknn!=3)               /* stop when find TYPES */
            {
            ocnt++;                     /* count and write outputs */
            pdelim(r2);
            getid(r1,33);
            find_token();
            }
         primt[sys_prims].outp=ocnt;
         features[sys_prims][0]=1;
         pdelim(r2);                   /* write the TYPES token   */
         getid(r1,33);                 /* get the next token      */
         find_token();
         if (toknn==4)                 /* this will happen if no  */
            {                          /* TYPES are declared      */
            count=0;
            fprintf(r2," ; \n");  /* puts TYPES: ; into file */
            pdelim(r2);             /* print the '{' */
            }
         else
            {
            while (toknn!=4)
               {                          /* write all the TYPES */
               pdelim(r2);
               getid(r1,33);
               find_token();
               }
            pdelim(r2);                /* then write the '{'  */
            }
         skip=0;
         while (1)                  /* get the rest of the circuit*/
           {
           ckt_line(&outcount,r1,outstack,&incount,instack,&skip);
           if (skip==1)
              break;
           for (i=0; i<outcount-1; i++)       /* write outputs*/
              fprintf(r2," ]s ,",outstack[i]);
           fprintf(r2," ]s = ",outstack[i]);
           fprintf(r2,"]s(",savfunc);  /*write the function name*/
           for (i=0; i<incount-1; i++)     /* write the inputs*/
              fprintf(r2," ]s ,",instack[i]);
           fprintf(r2," ]s ) ;\n",instack[i]);
           strcpy(token_buf,savfunc);
           findprim();                  /*see what func name is*/
           if (features[primid][1] > maxlevel) /*if higher level*/
```

```
                    maxlevel=features[primid][1];
                                /*then save higher level*/
                  }
               fprintf(r2," } \n");       /* write the closing brace */
               fclose(r2);
               features[sys_prims][1]=maxlevel+1;/*this prim is 1 level*/
               sys_prims++;              /* higher than highest subckt*  /
               r2=fopen("d:temp","r");
               if (r2==NULL)
                  printf("Temp file open failed\n");
               r3=fopen("d:struc","w");
               if (r3==NULL)
                  printf("Struc file open failed\n");
               fcopy(r2,r3);             /* copy new file back to */
               fclose(r2);               /* STRUC                 */
               fclose(r3);
               printf("writing PRIMITIV.DAT file\n");
               r4=fopen("d:primitiv.dat","w");
               if (r4==NULL)
                  printf("Primitive file open failed\n");
               fprintf(r4,"]d \n",sys_prims);
               for (i=0; i<sys_prims; i++)
                  {
                  fprintf(r4," ]s ]d ]d ]d ]d\n",
                        primt[i].nam2,primt[i].numpar,
                        primt[i].outp,features[i][0],features[i][1]);
                  }
               fclose(r4);
               getid(r1,33);
               find_token();
               break;
      case 28: printf("ADDBLOCK not installed yet....\n");
               break;
      } /*switch */
   } while (done==0);
print_select=0;
}/*addlib*/
 /*------------------END ADD_LIB-------------------------------*/
```

APPENDIX C
THE TIMING-WHEEL SIMULATOR PROGRAM

```
/*****************************************************************
 *                                                               *
 *                     Timing Wheel Program                      *
 *                          for the                              *
 *                  Multilevel Logic Simulator                   *
 *                                                               *
 *                     Version 3.1  27 Feb 87                    *
 *     Original Version by Ausif Mahmood at WSU for UNIX VAX     *
 *   Microcomputer versions and bug fixes by Scott Kelly at NPS  *
 *   Adapted to the version 3.1 of CADD program by Julio Cesar   *
 *   Lopes de Albuquerque at NPS. Use with CADD version 3.1.     *
 *****************************************************************/

int _mlen = 250 ;
int _mem[250] ;
#include "c:\lc\stdio.h"

/*
 This is the timing wheel program. First it initializes the circuit
 inter-connections in terms of descriptors (descriptor interconnec-
 tions are given in a file  "p2a"). The circuit is then  simulated
 according to the input data given in a file. Event directed simula-
 tion is used for maximum time efficiency -Ulrich's algorithm.
 Simulation results are directed to a file .  */

#define inputs 2            /*  2 inputs per descriptor allowed   */
#define maxprim 100         /*  maximum number of primitives      */
#define maxdescrpt 250     /*  maximum number of descriptors     */
#define maxsymb 50         /*  symbol table for printing         */
#define mdelay 100          /*  maximum possible delay            */
#define mdepth 250          /*  concurrent actions                */
#define maxmat 400          /*  delay matrix units                */
#define maxinput 100        /*  max number of inputs              */
#define maxtw 400           /*  timing wheel data structure       */
#define maxout 32           /*  maximum number of outputs per prim*/
/*---------------------------------------------------------------*/

struct matrix {             /* delay matrix for a primitive       */
  int rdelay0, rdelay1, fdelay0, fdelay1 ;
  struct matrix *matptr ;
  } ;
/*---------------------------------------------------------------*/

/*--------------RECORD ORGANIZATION FOR A DESCRIPTOR-------------*/
struct descrpt {
  int (*pfunc)() ;          /*  pfunc is pointer to the function  */
                            /*  i.e. code for the primitive       */
  int param[7] ;            /*  parameters for the function       */
                            /*  param[0] = input1 value           */
                            /*  param[1] = input2 value           */
                            /*  param[2] = rise1 delay            */
                            /*  param[3] = fall1 delay            */
                            /*  param[4] = rise2 delay            */
                            /*  param[5] = fall2 delay            */
                            /*  param[6] = MODE ( 0 = normal )    */
                            /*           ( 1 = uncertain if low ) */
                            /*           ( 2 = uncertain if high) */
                            /*           ( 3 = stuck-at-0 fault)  */
                            /*           ( 4 = stuck-at-1 fault)  */
  struct matrix *mptr ;     /* mptr is pointer to delay matrix    */
  struct descrpt *header ; /* header is a pointer to a descriptor*/
```

```
  struct descrpt *right0 ; /* 2 right pointers per descriptor    */
  struct descrpt *right1 ; /* block.                             */
  int h_value  ;           /* field indicator for header pointer */
  int r_value[inputs] ;    /* field indicators for right pointers*/
  int present_output  ;    /* present output of a primitive      */
  struct descrpt *ext_ptr ;/* extension pointer for multi-       */
                           /* descriptor primitives.             */
  int parent ;             /* parent descriptor number           */
 } ;
/*-------------------------------------------------------------*/

/*---------------------TIMING WHEEL STRUCTURES-----------------*/
 struct newstack {         /* new_values for multi-output        */
   int newvalue ;          /* functions attached to timing wheel */
   struct newstack *newptr;
 } ;
 struct time_stack {       /* timing_wheel structure             */
   struct descrpt *dptr ;  /* dptr = pointer to the descriptor   */
   int newval, sflag ;     /* newval= newvalue, sflag= scheduling*/
   struct newstack *nptr ; /* flag, nptr = pointer to newstack   */
   struct time_stack *tptr ;
 } ;
/*-------------------------------------------------------------*/

  FILE *rp           ;     /*  Read pointer to input data file   */
  FILE *wp           ;     /*  Write pointer to output data file */

/*----------------SYMBOL TABLE RECORD STRUCTURE----------------*/
  struct symb_tab {
    char name[8]  ;        /* name of the line                   */
    int index     ;        /* index = # of the associated desc.  */
                  } ;
/*-------------------------------------------------------------*/

/*----------------------GLOBAL PARAMETERS----------------------*/
  struct descrpt desc[maxdescrpt]; /*   STORAGE FOR STRUCTURES   */
  struct time_stack tstack[maxtw] ;
  struct newstack nstack[mdepth] ; /* timing wheel structures    */
  struct matrix matx[maxmat] ;
  struct symb_tab sym[maxsymb] ;
  struct descrpt *ptr, *parent_ptr ;
  struct newstack *nwptr, *fref, *npptr, *nw1pt ;
  struct time_stack *endt[mdelay], *begint[mdelay], *freft ;
  struct time_stack *savt, *fwdt, *tempt ;
  int num_outs ;                  /* # of outputs for a function  */
  int time, sav_time, sim_time ;
  int sav_write ;           /* write flag for printing output names*/
  int f_out[maxout] ;       /* outputs returned by each function   */
  int ff_out[maxout] ;
  int del[maxout], delay ;
  int timing_wheel, depth[mdelay] ;
/* depth[]  indicates the number of concurrent actions in 1 slot */
  int num_input, inpt[maxinput] , inptc[maxinput] ;
  int hashtable[100];       /* simple hashtable for variable names*/
  int hashcount;            /* number of items in hashtable       */
/*  num_input = number of inputs in the input data file          */
/*  inpt[] = array holding values of inputs                      */
/*  inptc[]= array holding hash value      of inputs             */
  int num_print , out_interval ;
  char tokenbuff[8] ;
  int endinputname ;
/* num_print  = number of lines to be printed out               */
/* out_interval=interval after which each output is to be printed*/
  int (*pnfn[maxprim])() ; /* pointer to primitive functions     */
  int pncnt ;
  int AND() ;
  int OR() ;
  int INVERT() ;
#include "c:\simd\ftype"
  int fmode() ;            /* mode behavior simulation */
```

```
  int fdel_ins() ;          /* finds delays applicable to a change*/
  int insert() ;            /* insert the desc. in proper slot    */
  int indata() ;            /* input data retrieval for descs.    */
  int read_input() ;
  int write_output() ;
  int hashf() ;
  int test();
  int getname() ;
  int fcopy() ;       /*  file copying routine                    */
/*-------------------------------------------------------------*/

/*************************************************************
 *                                                           *
 *                        MAIN PROGRAM                       *
 *                                                           *
 *************************************************************/
main(argc,argv)
int argc ;
char *argv[] ;
 {
 int i, j, k, fx, field_no, duminp[32] ;
 int flag, sav_value , savh;
 int sav_depth, endread, num1, num2, num3, num4 ;
 int filecount ;
 FILE *wq ;
 FILE *ti ;             /* pointer to initialization file    */
 FILE *tm ;             /* pointer to modif. delay file      */
 FILE *tc ;             /* pointer to descriptor file        */
 FILE *td ;             /* pointer to std. delay file        */
 FILE *tp ;             /* pointer to printout file          */
 struct descrpt *sav_ptr, *prev_ptr ;
 struct matrix *ptrm ;
 char z ;
/*-------------------------BEGIN-------------------------------*/

hashcount=0;
for (i=0; i<100; i++)
   hashtable[i] = -1;
 rp = fopen(argv[1],"r") ; /*   is the input data file      */
 pnfn[0] = read_input ;
 pnfn[1] = AND ;
 pnfn[2] = OR ;
 pnfn[5] = INVERT ;
#include "c:\simd\faddr"
 printf(" \n");
 printf("                                          \n");
 printf("    | MULTI-SIM version 1.0 -Nov. 1, 1984 |\n");
 printf("    ------------------------------------\n");
 printf(" \n");
 printf("                Loading Compiler Data...\n");
 printf(" \n");
/*----------------------INITIALIZE DELAY MATRIX----------------*/
 for (i = 0; i < maxmat; i = i + 1)
  {
   matx[i].rdelay0 = -1 ;
   matx[i].fdelay0 = -1 ;
   matx[i].rdelay1 = -1 ;
   matx[i].fdelay1 = -1 ;
   matx[i].matptr = NULL ;
  }
/*-------------------------------------------------------------*/
/*---------------------INITIALIZE DESCRIPTORS------------------*/
 for (i = 0; i < maxdescrpt ; i = i + 1 )
  {
   for (j = 2; j < 6 ;   j = j + 1 )
     desc[i].param[j] = -1 ;       /* initialize parameters to -1*/
   for (j = 0; j < inputs ; j = j + 1 )
     desc[i].r_value[j] = 0 ;
   desc[i].param[0] = 2 ;
   desc[i].param[1] = 2 ;          /* initialize inputs to 2's   */
```

```
   desc[i].param[6] = 0 ;          /* normal mode of operation   */
   desc[i].header = &(desc[i]) ;
   desc[i].h_value = 0 ;
   desc[i].parent = i ;
   desc[i].present_output = 2 ;    /*  2 stands for don't care   */
   desc[i].ext_ptr = NULL ;
   desc[i].mptr = NULL ;
  }
/*-----------------------------------------------------------------*/
/*--------------------Read input line names----------------------*/
 endinputname = 0 ;
 j = 0 ;
 while(1)
  {
   getname() ;
   inptc[j] = hashf(tokenbuff);
   j = j + 1 ;
   if (endinputname == 1)
     break ;
  }
 num_input = j ;
/*-----------------------------------------------------------------*/
/*---------INITIALIZE TIMING WHEEL & STORAGE POOL----------------*/
 for ( i = 0; i < mdelay ; i = i + 1 )
  {
   depth[i] = -1;
   beqint[i] = NULL ;
   endt[i] = NULL ;
  }
 j = maxtw - 1 ;
 for ( i = 0; i < j ; i = i + 1)
  {
   tstack[i].newval = 2 ;
   tstack[i].sflag  = 2 ;
   tstack[i].tptr = &(tstack[i+1]) ;
   tstack[i].nptr = NULL ;
  }
 freft = &(tstack[0]) ;
/*-----------------------------------------------------------------*/
/*-------STORAGE POOL FOR NSTACK(future multi-output values)-----*/
 j = mdepth - 1 ;
 for ( i = 0; i < j; i = i + 1)
   nstack[i].newptr  = &(nstack[i + 1])  ;
 fref = &(nstack[0]) ;  /* fref points to the first free nstack  */
/*-----------------------------------------------------------------*/
/*------------------INITIALIZE CIRCUIT CONNECTIONS---------------*/
     filecount = 0 ;
     wq = fopen("d:simdata","w");
     tm = fopen("d:modf","r");
     ti = fopen("d:initi","r");
     tc = fopen("d:descf","r");
     td = fopen("d:delf","r");
     tp = fopen("d:prnt","r");
     fcopy(tc,wq);
     fcopy(td,wq);
     fcopy(tm,wq) ;
     fcopy(ti,wq) ;
     fcopy(tp,wq);
     fclose(wq) ;
     fclose(tm) ;
     fclose(ti) ;
     fclose(tp) ;
     fclose(tc) ;
     fclose(td) ;
  wq = fopen("D:simdata","r") ;
  endread=0;
  while(endread==0)
   {
    fscanf(wq,"]d",&num1) ;
    switch (num1)
```

```
{
case 1 : fscanf(wq,"]d ]d ]d",&num2, &num3, &num4) ;
         switch (num3)
          {
           case 8 : desc[num2].header = &(desc[num4]);
                    break ;
           case 9 : if (num4 == 0)
                      desc[num2].right0 = ptr ;
                    else
                      desc[num2].right1 = ptr ;
                    break ;
           case 11 :desc[num2].h_value = num4 ;
                    break ;
           case 12: desc[num2].r_value[num4] = savh ;
                    break ;
           case 15: desc[num2].ext_ptr = &(desc[num4]) ;
                    break ;
           case 16: desc[num2].parent = num4 ;
                    break ;
           default: if (num3 < 7)
                      desc[num2].param[num3] = num4 ;
                    else
                     {
                      printf(" error in decoding code\n") ;
                       endread = 1 ;
                      break ;
                     }
          }
         break ;
case 2 : fscanf(wq,"]d",&num2) ;
         savh = desc[num2].h_value ;
         break ;
case 3 : fscanf(wq,"]d",&num2) ;
         ptr = desc[num2].header ;
         break ;
case 4 : fscanf(wq,"]d",&num2) ;
         ptr = &(desc[num2]) ;
         break ;
case 5 : fscanf(wq,"]d ]d",&num2, &num3) ;
         (*ptr).param[num2] = num3 ;
         break ;
case 6 : fscanf(wq,"]d",&num2) ;
         ptr = desc[num2].ext_ptr ;
         break ;
case 7 : ptr = (*ptr).ext_ptr ;
         break ;
case 8 : ptrm = (*ptr).mptr ;
         break ;
case 9 : ptrm = (*ptrm).matptr ;
         break ;
case 10: fscanf(wq,"]d",&num2) ;
         (*ptrm).rdelay0 = num2 ;
         break ;
case 11: fscanf(wq,"]d",&num2) ;
         (*ptrm).fdelay0 = num2 ;
         break ;
case 12: fscanf(wq,"]d",&num2) ;
         (*ptrm).rdelay1 = num2 ;
         break ;
case 13: fscanf(wq,"]d",&num2) ;
         (*ptrm).fdelay1 = num2 ;
         break ;
case 14: fscanf(wq,"]d",&num2) ;
         (*ptr).mptr = &(matx[num2]) ;
         break ;
case 15: fscanf(wq,"]d ]d",&num2, &num3) ;
         matx[num2].matptr = &(matx[num3]) ;
         break ;
case 16: fscanf(wq,"]d ]d",&num2, &num3) ;
         matx[num2].rdelay0 = num3 ;
```

```
                break ;
        case 17: fscanf(wq,"]d ]d",&num2, &num3) ;
                matx[num2].fdelay0 = num3 ;
                break ;
        case 18: fscanf(wq,"]d ]d",&num2, &num3) ;
                matx[num2].rdelay1 = num3 ;
                break ;
        case 19: fscanf(wq,"]d ]d",&num2, &num3) ;
                matx[num2].fdelay1 = num3 ;
                break ;
        case 20: depth[0] = depth[0] + 1 ;
                break ;
        case 21: begint[0] = freft ;
                break ;
        case 22: (*(endt[0])).tptr = freft ;
                break ;
        case 23: endt[0] = freft ;
                break ;
        case 24: freft = (*freft).tptr ;
                break ;
        case 25: (*(endt[0])).tptr = NULL ;
                break ;
        case 26: fscanf(wq,"]d",&num2) ;
                (*(endt[0])).dptr = &(desc[num2]) ;
                break ;
        case 27: fscanf(wq,"]d",&num2) ;
                (*(endt[0])).newval = num2 ;
                break ;
        case 28: fscanf(wq,"]d ]d",&num2, &num3) ;
               /*     fscanf(wq, "]c",&z) ;     if output names */
                z=getc(wq);
                z=getc(wq);
                      /* mess up                                  */
                sym[num2].name[num3] = z ;
                break ;
        case 29: fscanf(wq,"]d ]d",&num2, &num3) ;
                sym[num2].index = num3 ;
                break ;
        case 30: sav_write = 0 ;
                break ;
        case 31: fscanf(wq,"]d",&num2) ;
                num_print = num2 ;
                break ;
        case 32: out_interval = 1 ;
                break ;
        case 33: fscanf(wq,"]d ]d",&num2, &num3) ;
                desc[num2].pfunc = pnfn[num3] ;
                break ;
        case 50: endread = 1 ;
                break ;
        default: printf(" error in input data decoding\n") ;
      }
  }     /* end while */
fclose(wq) ;
/*-------------------------------------------------------------------*/
 wp = fopen(argv[2],"w") ; /*   is the output data file     */
                           /* connections                           */
 printf("        Welcome to MultiSimPC \n");
 printf(" \n");
 time = -1 ;
 timing_wheel = -1 ;
 sav_time = 0 ;
 printf(" Please enter Simulation time: ") ;
 scanf("]d",&sim_time) ;
 printf(" \n") ;
/*------------INITIALIZE INPUT DESCS. IN TIMING WHEEL------------*/
 for ( i = 0; i < num_input; i = i + 1 )
  {
   ptr = &(desc[i]) ;      /* beginning descs. = input descs.*/
   (*((*ptr).pfunc))(i) ;             /* call to read_input */
```

```
  }
/*-------------------------------------------------------------*/
 timing_wheel = 0 ;
/*-------------------------------------------------------------*/
/*--------------------BEGIN TIMING WHEEL-----------------------*/
 while ( time <= sim_time )
  {                              /* begin while time < sim time */
   if (timing_wheel >= mdelay)
     timing_wheel = 0 ;          /* Timing wheel is circular    */
   while ( timing_wheel <  mdelay )
    {                           /* begin 1 loop of timing_wheel */
     time = time + 1 ;
     if (time > sim_time)
       break ;
     sav_depth = depth[timing_wheel] ;
     savt = begint[timing_wheel] ;        /* ptr to first element*/
     while ( depth[timing_wheel] != -1)
      {                /* while all row slots have been updated  */
       /*-----------------BEGIN UPDATE-------------------------*/
       /* All descriptors connected to the current descriptor are*/
       /* updated with the 'future value' from the timing wheel  */
       /* if the present value differs from the 'future value'   */
       fwdt = begint[timing_wheel] ;
       ptr = (*fwdt).dptr ;
       begint[timing_wheel] = (*(begint[timing_wheel])).tptr ;
       /* begint points to the next element in the current row   */
       /*-------------SCHEDULE INPUT READING--------------------*/
       if (((*fwdt).sflag) == 1)
        {     /* if scheduling flag is 1, schedule the descriptor */
         (*((*ptr).pfunc))((*ptr).parent) ;
         (*fwdt).sflag = 2 ;
                                    /* de-assert scheduling flag */
        }
       /*-------------------------------------------------------*/
       depth[timing_wheel] = depth[timing_wheel] - 1 ;
       flag = 0 ;      /* flag = 1 indicates a multi-output desc. */
       sav_value = (*fwdt).newval ;
                 /* sav_value = future value from tim. wheel     */
                 /* i.e. value to be replaced for present output */
       while (1)
        {        /* begin while C-list for each output is updated */
         sav_ptr = ptr ;
         if (sav_value != -1)
          {
           if (((*ptr).present_output) != sav_value)
            {  /* change has occured */
             ptr = (*ptr).header ;
             if (ptr != sav_ptr)
              {    /* if not end of circular list, then begin     */
               (*ptr).param[(*sav_ptr).h_value] = sav_value ;
                /* input no. pointed to by the desc. is updated  */
               prev_ptr = ptr ;
               switch ((*sav_ptr).h_value)
                {
                 case 0 : ptr = (*ptr).right0 ;
                          field_no = (*prev_ptr).r_value[0] ;
                          break ;
                 case 1 : ptr = (*ptr).right1 ;
                          field_no = (*prev_ptr).r_value[1] ;
                }
               while (1)
                { /* while begin */
                 if (ptr == sav_ptr)
                 /* if end of circular list, then get out */
                   break ;
                 (*ptr).param[field_no] = sav_value ;
                 /* update input with 'future value' */
                 prev_ptr = ptr ;
                 switch (field_no)
                  {
```

```
                case 0 : ptr = (*ptr).right0 ;
                         fx = (*prev_ptr).r_value[0] ;
                         break ;
                case 1 : ptr = (*ptr).right1 ;
                         fx = (*prev_ptr).r_value[1] ;
                }
              field_no = fx ;
            } /*     end while                          */
          }   /*     end then (if not end of C-list)    */
        }     /*     end if change has occured          */
      }       /*     if sav_value != -1                 */
    if (flag == 1)
     {
      if ((*nwptr).newptr == NULL)
        break ;          /* if nstack has no more extension */
      nwptr = (*nwptr).newptr ;
     }
    else
     {
      if ((*fwdt).nptr == NULL)
        break ;
      nwptr = (*fwdt).nptr ;
      flag = 1 ;
           /* flag is asserted for a multi-output function */
     }
    ptr = (*sav_ptr).ext_ptr ;      /* if multi-output case */
    sav_value = (*nwptr).newvalue ;
   }         /* end update of all descs. in C-list */
 }           /* end of update of one row of T.W.   */
 /*--------------END UPDATE-------------------------------*/
depth[timing_wheel] = sav_depth ;
begint[timing_wheel] = savt ;
/*------------------EXECUTION PHASE----------------------*/
/* Second pass -Schedule the descs. in C-list & insert in   */
/* proper time slot if output of current desc. has changed  */
while ( depth[timing_wheel] != -1)
 {          /* begin execution of one row of timing wheel  */
  fwdt = begint[timing_wheel] ;
  begint[timing_wheel] = (*(begint[timing_wheel])).tptr ;
  ptr = (*fwdt).dptr ;
  depth[timing_wheel] = depth[timing_wheel] - 1 ;
  sav_value = (*fwdt).newval ;
              /* future value */
  flag = 0 ;       /* flag = 1 indicates multi-output desc. */
  while (1)
   {      /* begin while not end of C-list for all outputs */
    sav_ptr = ptr ;
    if (sav_value != -1)
     {
      if (((*ptr).present_output) != sav_value)
       {                              /* change has occured */
        (*ptr).present_output = sav_value ;
                                  /* update present output */
        ptr = (*ptr).header ;
        if (ptr != sav_ptr)
         {         /* if not end of C-list, then begin      */
          /*---SCHEDULE DESC. ----------------------------*/
          parent_ptr = &(desc[(*ptr).parent]) ;
          (*((*parent_ptr).pfunc))(0,duminp,f_out) ;
          /*------------------------------------------------*/
          fmode() ;     /* mode simulation */
          fdel_ins((*sav_ptr).h_value);
                    /* find delays and insert in proper slot */
          prev_ptr = ptr ;
          switch ((*sav_ptr).h_value)
           {
            case 0 : ptr = (*ptr).right0 ;
                     field_no = (*prev_ptr).r_value[0] ;
                     break ;
            case 1 : ptr = (*ptr).right1 ;
```

```
                    field_no = (*prev_ptr).r_value[1] ;
                }
              while (1)
                { /* while begin */
                 if (ptr == sav_ptr) /* if end of C-list, quit   */
                   break ;
                 /*---SCHEDULE DESC. -get all parameters first --*/
                 parent_ptr = &(desc[(*ptr).parent]) ;
                 (*((*parent_ptr).pfunc))(0,duminp,f_out) ;
                 /*-----------------------------------------*/
                 fmode() ;       /* simulate mode behavior */
                 fdel_ins(field_no) ;
                       /* find delays and insert in proper slot */
                 prev_ptr = ptr ;
                 switch (field_no)
                  {
                   case 0 : ptr = (*ptr).right0 ;
                            fx = (*prev_ptr).r_value[0] ;
                            break ;
                   case 1 : ptr = (*ptr).right1 ;
                            fx = (*prev_ptr).r_value[1] ;
                  }
                 field_no = fx ;
                } /*     end while                          */
              }   /*     end then  (if not end of C-list) */
            }     /*  end if change has occured           */
          }       /*  end if sav_value != -1              */
        if (flag == 1)       /* multi output case        */
          {
           if ((*nwptr).newptr == NULL)
             {
             /* fref storage for nstack */
              (*nwptr).newptr = fref ;
              fref = nwptr ;
             break ;  /* If all outputs have been handled, quit */
             }
           /* free storage for the previous nstack */
           npptr = nwptr ;
           nwptr = (*nwptr).newptr ;
           (*npptr).newptr = fref ;
           fref = npptr ;
          }
        else     /* if flag is 0 */
          {
           if ((*fwdt).nptr == NULL)
             break ;
           nwptr = (*fwdt).nptr ;
           flag = 1 ;     /* multi-output case   */
          }
         ptr = (*sav_ptr).ext_ptr ;
         sav_value = (*nwptr).newvalue ;
        }   /* end (C-list scan for all outputs) */
       /*-------------Free storage for current tstack ----------*/
       tempt = freft ;
       freft = fwdt ;
       (*freft).tptr = tempt ;
       /*-----------------------------------------------------*/
      }     /* end of one row of timing wheel    */
     write_output() ;       /* print results */
     begint[timing_wheel] = NULL ;
     endt[timing_wheel] = NULL ;
     timing_wheel = timing_wheel + 1 ;
                          /* move to next row of timing wheel */
    }       /* end of 1 loop (vertical) of timing wheel */
   }        /* end time < sim time                      */
  fclose(rp) ;
  fclose(wp) ;
 }  /* end of main program */
/*------------------END OF MAIN -------------------------------*/
```

```
/*****************************************************************
 *                                                               *
 *                        FMODE SUBROUTINE                       *
 *                                                               *
 *****************************************************************/
/* It finds outputs for each descriptor according to its mode    */
fmode()
 {
  int i, mode ;
  struct descrpt *xtptr ;
  xtptr = parent_ptr ;
  for (i = 0; i <  num_outs ; i = i + 1)
   {
    mode = (*xtptr).param[6] ;
    switch (mode)
     {
      case 0 : f_out[i] = f_out[i] ;
               break;
      case 1 : if (f_out[i] == 0)
                 f_out[i] = 2 ;
               break;
      case 2 : if (f_out[i] == 1)
                 f_out[i] = 2 ;
               break;
      case 3 : f_out[i] = 0 ;
               break ;
      case 4 : f_out[i] = 1 ;
     }
    xtptr = (*xtptr).ext_ptr ;
   }
 }
/*------------------END FMODE-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                     FDEL_INS SUBROUTINE                       *
 *                                                               *
 *****************************************************************/
/* This procedures finds the delays applicable to a function     */
/* connected to the output of the current descriptor. The input  */
/*number to which the current descriptor is connected is supplied*/

fdel_ins(inp)
int inp ;        /* inp = input number to which the current desc.*/
 {               /* connected  */
  int i, j, k ;
  struct matrix *mtptr ;
/*----------------------COMPUTE DELAYS--------------------------*/
  if (f_out[0] == 1)
   {
    if (inp == 0)
      del[0] = (*ptr).param[2] ;        /* del-0 = rise del(0,0) */
    else
      del[0] = (*ptr).param[4] ;        /* del-0 = rise del(1,0) */
   }
  else    /* if first output is 1 */
   {
    if (inp == 0)
     del[0] = (*ptr).param[3] ;         /* del-0 = fall del(0,0) */
    else
     del[0] = (*ptr).param[5] ;         /* del-0 = fall del(1,0) */
   }
  if ((*ptr).mptr != NULL)             /* multi-output case     */
   {
    mtptr = (*ptr).mptr ;
    for (i = 1; i < num_outs; i = i + 1)
     {
      if (f_out[i] == 1)
       {
        if (inp == 0)
```

```
          del[i] = (*mtptr).rdelay0 ;   /* del-i = rise del(0,i) */
        else
          del[i] = (*mtptr).rdelay1 ;   /* del-i = rise del(1,i) */
       }
      else    /* if ith output is 0 */
       {
        if (inp == 0)
          del[i] = (*mtptr).fdelay0 ;   /* del-i = rise del(0,i) */
        else
          del[i] = (*mtptr).fdelay1 ;   /* del-i = fall del(1,i) */
       }
      if ((*mtptr).matptr == NULL)
        break ;
      else
        mtptr = (*mtptr).matptr ;
     }
   }
/*-----------------END COMPUTE DELAYS----------------------------*/
/* For a multi-output function, different delays are possible.   */
/* Insert in proper slot for each different delay                */
  for (i = 0; i < num_outs; i = i + 1)
   {
    if (del[i] != -1)   /* delay = -1 means input and output are */
     {                  /* not related.                          */
      delay = del[i] ;
      ff_out[i] = f_out[i] ;
      /*-------------------Check for identical delays -----------*/
      for (j = i+1; j < num_outs; j = j + 1)
       {
        if (del[j] == delay)      /* For identical delays on     */
          ff_out[j] = f_out[j] ;  /* different outputs, function */
        else                      /* should be inserted only once*/
          ff_out[j] = -1 ;        /* -1 output means, keep the   */
       }                          /* output                      */
       /*-----------------------------------------------------*/
      /*-------Eliminate the delay cases which have been covered */
      for (k = 0; k < num_outs; k = k + 1)
       {
        if (del[k] == delay)
          del[k] = -1 ;
       }
       /*-----------------------------------------------------*/
      insert() ;       /* insert in proper slot */
     }
    else               /* if delay = -1 */
      ff_out[i] = -1 ;
   }
 }
/*------------------END FDEL_INS-------------------------------*/

/*****************************************************************
 *                                                               *
 *                     INSERT  SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* This procedure inserts the function in proper slot in T.W.    */

insert()
 {
  int i,j,k, slot_no ;
  slot_no = (delay + timing_wheel)]mdelay ;
  depth[slot_no] = depth[slot_no] + 1 ;
  if (endt[slot_no] != NULL)
    (*(endt[slot_no])).tptr = freft ;
  else
    begint[slot_no] = freft ;
/* allocate storage for tstack */
  endt[slot_no] = freft ;
  freft = (*freft).tptr ;
  (*(endt[slot_no])).tptr = NULL ;
```

```
/*-----------------------------*/
  (*(endt[slot_no])).dptr = parent_ptr ;
  (*(endt[slot_no])).newval = ff_out[0] ;
  if (num_outs > 1)       /* multi-output cases */
   {
    (*(endt[slot_no])).nptr = fref ;
    /* allocate storage for newstack */
    (*fref).newvalue = ff_out[1] ;
    nwlpt = fref ;
    fref = (*fref).newptr ;
    if (num_outs > 2)
     {
      k = 0 ;
      j = 2 ;
      while (k == 0)
       {
        (*nwlpt).newptr = fref ;
        nwlpt = fref ;
        (*fref).newvalue = ff_out[j] ;
        fref = (*fref).newptr ;
        j = j + 1 ;
        if (j == num_outs)
          break ;
       }
     }
    (*nwlpt).newptr = NULL ;
   }
  else        /* if single output function */
   (*(endt[slot_no])).nptr = NULL ;
              /* put new value in the slot */
 }
/*------------------END INSERT---------------------------------*/

/*****************************************************************
 *                                                               *
 *                    READ_INPUT SUBROUTINE                      *
 *                                                               *
 *****************************************************************/
read_input(ddnum)
int ddnum ;
{
 int i, order, k, slotn, p0, p2 ;
/*  determine input order i.e. which input is to be read  */
 p0 = desc[ddnum].param[0] ;
 p2 = desc[ddnum].param[2] ;
 for ( i = 0; i < num_input ; i = i + 1 )
  { if ( p0 == inptc[i]  )
      break ;
   }
 order = i ;
                  /* input order found so quit searching */
 if (sav_time != time )
  {
   sav_time = time ;
   for ( i = 0; i < num_input ; i = i + 1 )
    {
     fscanf(rp,"]d",&k)  ;
     inpt[i] = k ;
    }
  }
 f_out[0] = inpt[order] ;
 slotn = (timing_wheel + p2)](mdelay) ;
 depth[slotn] = depth[slotn] + 1 ;
 if (endt[slotn] != NULL )
   (*(endt[slotn])).tptr = freft ;
 else
   begint[slotn] = freft ;
/* allocate storage for tstack */
 endt[slotn] = freft ;
 freft = (*freft).tptr ;
```

```
 (*(endt[slotn])).tptr = NULL ;
/*---------------------------*/
 (*(endt[slotn])).dptr = &(desc[order]) ;
 (*(endt[slotn])).newval = f_out[0] ;
 (*(endt[slotn])).nptr = NULL ;
 (*(endt[slotn])).sflag = 1 ;
}
/*------------------END READ_INPUT-----------------------------*/

/*****************************************************************
 *                                                               *
 *                WRITE_OUTPUT SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
write_output ()
/* num_print  contains the number of lines to be printed        */
/* out_interval is interval after which value is to be printed  */
/* sym[].index contains indices of descs. representing output    */
{
 int i ;
 if (sav_write == 0)
  {
   fprintf(wp," time");
   for (i = 0; i < num_print ; i = i + 1)
     fprintf(wp," ]5s",sym[i].name) ;
   fprintf(wp,"\n") ;
   fflush(wp) ;
  }
 sav_write = 1 ;
   fprintf(wp, " ]3d ",time) ;
 for (i = 0; i < num_print ; i = i + 1)
   fprintf(wp," ]5d",desc[(sym[i].index)].present_output) ;
 fprintf(wp,"\n") ;
 fflush(wp) ;
}
/*------------------END WRITE_OUTPUT---------------------------*/

#include "c:\simd\block"

/*****************************************************************
 *                                                               *
 *                        INVERT SUBROUTINE                      *
 *                                                               *
 *****************************************************************/
INVERT(flg,  inpx, ou)
int flg , *inpx, *ou ;
 {
  if (flg == 0)
    indata(inpx, 1) ;
  switch((*inpx))
   {
   case 0 : (*ou) = 1 ;
            break ;
   case 1 : (*ou) = 0 ;
            break ;
   case 2 : (*ou) = 2 ;
   }
 num_outs = 1 ;
 }
/*------------------END INVERT---------------------------------*/

/*****************************************************************
 *                                                               *
 *                          AND SUBROUTINE                       *
 *                                                               *
 *****************************************************************/
 AND(flg , inpx, ou)
 int flg , *inpx, *ou ;
 {
  int s ;
```

```
  if (flg == 0)
    indata(inpx, 2) ;
  s = (*inpx) + (*(inpx + 1)) ;
  switch(s)
   {
    case 0 : (*ou) = 0 ;
             break ;
    case 1 : (*ou) = 0 ;
             break ;
    case 2 : if ((*inpx) == 1)
               (*ou) = 1 ;
             else
               (*ou) = 0 ;
             break;
     default : (*ou) = 2 ;
    }
   num_outs = 1 ;
  }
/*-------------------END AND-------------------------------------*/

/*****************************************************************
 *                                                               *
 *                          OR  SUBROUTINE                       *
 *                                                               *
 *****************************************************************/
 OR(flg ,inpx, ou)
 int flg , *inpx, *ou ;
 {
  int s ;
  if (flg == 0)
    indata(inpx, 2 ) ;
  s = (*inpx) + (*(inpx+1)) ;
  switch(s)
   {
    case 0 : (*ou) = 0 ;     /* ou is pointer to f_out */
             break ;
    case 1 : (*ou) = 1 ;
             break ;
    case 2 : if ((*inpx) == 1)
               (*ou) = 1 ;
             else
               (*ou) = 2 ;
             break;
     case 3 : (*ou) = 1 ;
             break ;
     default : (*ou) = 2 ;
    }
   num_outs = 1 ;
  }
/*-------------------END OR--------------------------------------*/

/*****************************************************************
 *                                                               *
 *                       INDATA SUBROUTINE                       *
 *                                                               *
 *****************************************************************/
/* input data retrieval for a function                          */
 indata(inpx,inns)
 int *inpx, inns ;
 /* inpx = input data array,   inns = number of input data       */
  {
   struct descrpt *tmptr ;
   int i ;
   if (inns >= 1)
    {
     (*inpx) = (*parent_ptr).param[0] ;
     inns = inns - 1 ;
     if (inns != 0)
      {
       (*(inpx+1)) = (*parent_ptr).param[1] ;
```

```
       inns = inns - 1 ;
      }
     i = 2 ;
     tmptr = parent_ptr ;
     while (inns != 0)
      {
       tmptr = (*tmptr).ext_ptr ;
       (*(inpx + i))= (*tmptr).param[0] ;
       inns = inns - 1;
       i = i + 1 ;
       if (inns != 0)
        {
         (*(inpx + i)) = (*tmptr).param[1] ;
         i = i + 1 ;
         inns = inns - 1 ;
        }
      }     /* end while */
    }
  }
/*------------------END INDATA-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                        HASHF SUBROUTINE                       *
 *                                                               *
 *****************************************************************/
hashf(s)    /* forms hash value for string s */
char *s;
{
  int hashval ;
  for (hashval = 0;  *s != '\0' ; )
     hashval += *s++ ;                /* hash ID name into an index */
  test(&hashval);                     /* and test for collisions    */
  hashtable[hashcount]=hashval;
  hashcount++;

  return (hashval) ;
}
/*------------------END HASHF------------------------------------*/

/*****************************************************************
 *                                                               *
 *                         TEST SUBROUTINE                       *
 *                                                               *
 *****************************************************************/
test(value)
int *value;
   {
   int i;
   for (i=0; i<hashcount; i++)
      {
      if (hashtable[i]==(*value))        /* collision?    */
         {
         (*value)=(*value)+11;           /* yes, add a prime number */
         test(value);                    /* and test this one */
         }
      }
   }
/*------------------END TEST-------------------------------------*/

/*****************************************************************
 *                                                               *
 *                      GETNAME SUBROUTINE                       *
 *                                                               *
 *****************************************************************/
getname()   /* returns the name in tokenbuff */
{
 int i, c , flag, delimiter ;
 i = 0 ;
 delimiter = -1 ;
```

```
flag = 0 ;
while(( delimiter < 1) || (flag == 0))
 {
  c = fgetc(rp) ;
  switch (c)
   {
    case ' ' : delimiter = 1 ;
               break ;
    case ',' : delimiter = 1 ;
               break ;
    case '\n': delimiter = 1 ;
               endinputname = 1 ;
               flag = 1 ;
               break ;
    default : flag = 1 ;
              delimiter = 0 ;
   }
  if (delimiter == 0)
   {
     if (i <= 6)
      {
       tokenbuff[i] = c ;
       i = i + 1 ;
      }
   }
 }
 tokenbuff[i] = '\0' ;
}
/*-------------------END GETNAME---------------------------------*/

/*****************************************************************
 *                                                               *
 *                    FCOPY  SUBROUTINE                          *
 *                                                               *
 *****************************************************************/
fcopy(rr, ww)
FILE *rr, *ww ;
{
 int c ;
 while ((c = getc(rr)) != EOF)
   putc(c,ww) ;
}
/*-------------------END FCOPY-----------------------------------*/
```

APPENDIX D
THE EDITOR PROGRAM

```
/*****************************************************************
 *                                                               *
 *                     Edition Program                           *
 *                         for the                               *
 *                   Multilevel Simulator                        *
 *                                                               *
 *                 VERSION 3.1, 14 Apr 1987.                     *
 *     Original version developed under MSDOS and PCDOS by       *
 *     Julio Cesar Lopes de Albuquerque at Naval Postgraduate    *
 *     School. Use with CADD version 3.1.                        *
 *                                                               *
 *****************************************************************/

int _mlen = 500;
int _mem[500] ;
#include "\lc\stdio.h"      /*<stdio.h> in DOS 3.x environment   */

#define maxkey 30             /* maximum number of keywords      */
#define maxsym 500            /* symbol table size               */
                              /* 1000 size in UNIX               */
#define maxprim 100           /* maximum number of primitives    */
#define maxouts 32            /* maximum of 32 outputs per prim. */
#define maxnorm 50            /* normload table size             */
#define maxk 14               /* user options for the program    */

/*------------------GLOBAL DECLARATIONS-------------------------*/
extern int strcpy() ;       /*  copies one string to another       */
extern int primsetup();     /*  external module to name the prims  */
extern int strcmp() ;       /*  string comparison                  */
extern int getid()  ;       /*  get next identifier                */
extern int find_token() ;   /*  returns the token number           */
                            /*  (keyword table index)              */
extern int IDSTRING() ;     /*  list of id's parsing               */
extern int rfdel() ;        /*  rise/fall delay handling           */
int bdread() ;              /*  block delay reading routine        */
extern int cmode() ;        /*  code generation for mode           */
extern int matgen() ;       /*  delay matrix and mode gen.         */
extern int parseid() ;      /*  single id parsing                  */
extern int findid() ;       /*  finds symbol table index           */
extern int idout() ;        /*  verify the output of the gate      */
extern int idinp() ;        /*  verify the inputs of the gate      */
int repl() ;                /*  replace any gate in the circuit    */
int cp_sim();               /*  copy a file until a desired point  */
int upfan();                /*  update fanout                      */
int mdfyfan();              /*  modify fanout                      */
int new_sim() ;             /*  copy the SIMDATA from a known      */
                            /*  point until the end                */
extern int prim() ;         /*  verify what primitive will be used */
extern int find_key() ;     /*  select the option of the user      */
int finddesc() ;            /*  finds symbol table index for       */
                            /*  the given function name/type       */
extern int updesct() ;      /*  updates the descriptor table       */
                            /*  (name and symbol table index)      */
extern int findprim() ;     /*  finds primitive library index      */
extern int update() ;       /*  update symbol table                */
extern int connect() ;      /*  code gen. and fanld update         */
                            /*  (descriptor interconnections)      */
int dlt_del();              /*  manages the deletion of delays     */
int mdy_pri();              /*  modify the printout                */
int copy_sim();             /*  copy part of a file                */
extern int code_input() ;   /*  code generation for INPUTS         */
```

```
                                /*  declaration                          */
extern int errmessage() ;       /*  error message printing               */
extern int outerror() ;         /*  output error message                 */
extern int error() ;            /*  error handling routine               */
extern int fcopy() ;            /*  file copying routine                 */
extern int fadvance() ;         /*  advances to next line                */
extern int hashf() ;            /*  converts input name to number        */
extern int test();              /*  tests for hash collisions            */
int chgdel();                   /*  used to change the delay part        */
int fnddel();                   /*  used to find a delay in the file     */
int chgend();                   /*  change the print part of a file      */
int ersprnt();                  /* erase part of the printout            */
int insgate();                  /*  used for insert a gate               */
int chginp() ;                  /*  used to change an input              */
int delgate();                  /*  used to delete a gate                */
int ersinp() ;                  /*  used to erase a input                */
/*-------------------------------------------------------------------*/

/*----------------DATA STRUCTURES------------------------------------*/
struct sym_tab {                /*  symbol table                         */
  char name[8] ;                /*  name = name of id                    */
  int descno, funcno ;          /*  descno = descriptor number           */
                                /*  funcno = primitive lib index         */
  int fanld;                    /*  fanld = actual circuit load          */
  int despos, delpos;           /*  despos = descriptor spaces in file   */
                                /*  delpos = delay spaces in the file    */
  int ini_num, pri_num;         /*  ini_num = initialization order       */
                                /*  pri_num = printout order             */
  int pri_val;                  /*  pri_val = # characters in variable   */
 } ;

struct desc_tab {               /*  table containing function            */
  char fun[8] ;                 /*  names (type names) and their         */
  int dnum ;                    /*  symbol table indexes                 */
 } ;

struct norm_tab {               /*  table containing function            */
  char nom[8] ;                 /*  names/types and associated           */
  int nmld ;                    /*  normload declared in DEFINE          */
 } ;

struct prim_tab {               /*  Primitive table :                    */
  char nam[8] ;                 /*  primitive table                      */
  char nam2[8] ;                /*  EXTENDED primitive table             */
  int numpar, outp ;            /*  numpar = no. of parameters           */
  int normld, fanout ;          /*  for the function                     */
  int technology, overld ;      /*  outp = # of outputs                  */
 } ;

struct err_stack {              /* stack for errors in one line          */
  char nm[8] ;                  /* nm = name of unexpected id            */
  int errno ;                   /* errno = error number                  */
 } ;

struct inp_name {               /* holds all the inputs for each gate    */
  char iname[8];                /* iname = name of the variable          */
  int inp_num;                  /* inp_num = # of inputs in the gate     */
  char inp1[8], inp2[8];        /* inp1 to inp10 = inputs for the gate   */
  char inp3[8], inp4[8];
  char inp5[8], inp6[8];
  char inp7[8], inp8[8];
  char inp9[8], inp10[8];
  int ifin;                     /* ifin = termination of the table       */
  };

struct tab_del {          /* holds all the information about the       */
  int indx, dsc_nb, typ_num;  /* gates that have modified delays       */
  int num_ipt, num_opt, val;
  };
/*-------------------------------------------------------------------*/
```

```
/*---------------STORAGE ALLOCATION-------------------------------*/
struct sym_tab symt[maxsym] ;
struct desc_tab desct[maxsym] ;
struct norm_tab nort[maxnorm] ;
struct prim_tab primt[maxprim], *primptr ;
struct err_stack errt[5] ;      /* max of 5 errors per line      */
struct inp_name inptab[maxsym];
struct tab_del del_tab[100];

int err_ptr ;                  /* error table pointer (count)    */
                               /* for one line.                  */
int matcount ;                 /* delay matrix count             */
int delimiter, bb ;            /* delimiter = delimiter type     */
                               /* bb = buff[80] index (line)     */
int rdmat[maxouts][maxouts], fdmat[maxouts][maxouts] ;
                               /* rise and fall delay matrices   */
int toknn, err_count ;         /* err_count  = error count       */
int features[maxprim][2];
int desc_no , sym_count, symid ;   /* desc_no = desc. count      */
int dptr, descid, lim ;        /* descriptor table indices       */
                               /* dptr = descriptor table cnt    */
int normcount ;                /* normtable count                */
int filecount;                 /* poutcount used for debugging   */
int prinpflag;                 /* controls printing of source    */
int end, outpar;
int prim_count, primid ;       /* prim_count = # of primitives   */
int sys_prims ;                /* number of permanent (system)   */
                               /* primitives                     */
int savprim ;                  /* save the primitive to be used  */
char token_buf[8], savbuf[8], buff[80] ; /* buff = 1 line        */
char keyword[maxkey][8] ;      /* keyword table                  */
char key[maxk][8];             /* key table                      */
int hashtable[100];              /* the hash table               */
int hashcount;                   /* number of items in hashtable */
int out_sub, old_prim;           /* holds the primitives in REPL */
int dct_del, olddel_sym ;
int savid[maxouts] ;
int old_desc, new_prim, num1, num2;
int old_val, old_ipt, old_del, desc_old;
int num3, iout;
int index, index1, old_func;
int ord_ini, ord_pri, endf;
int val_sym, dpt1 ;
int sym1, valact ;
int skp5, no_new ;
char z;
int inum, ant_desc;
int target, temp1, par1;
int parm1, par2, parm2;
int savpar1, savpar2, num;
int savtoken;
int del_sym, del_ipt, del_dct;        /* used in the delete case  */
int new, numb_inp, skp;
int del_val, skp1, skp2;
int savn[maxouts];
int tot_val , old_ptr , old_sym;
int skpd, skpm, skpi, skpp ;
int posit ,skpc, skp3, skp4 ;
int val_prt ;
char savfunc[8], userprg[8];

FILE *r1 ;              /* pointer to input data file        */
FILE *t1 ;              /* pointer to temp file              */
FILE *t2 ;              /* pointer to temp2 file             */
FILE *t3 ;              /* pointer to temp3 file             */
FILE *t4 ;              /* pointer to temp4 file             */
FILE *t5 ;              /* pointer to temp5 file             */
FILE *t6 ;              /* pointer to temp6 file             */
FILE *ti ;              /* pointer to initialization file    */
```

```
FILE *tm ;              /* pointer to modif. delay file      */
FILE *tc ;              /* pointer to descriptor file        */
FILE *td ;              /* pointer to std. delay file        */
FILE *tp ;              /* pointer to printout file          */
FILE *rp ;              /* read pointer to input data file   */
FILE *sy ;              /* symt table stored for edition     */
FILE *de ;              /* desct table stored for edition    */
FILE *nm ;              /* nort table stored for edition     */
FILE *ip ;              /* input table stored for edition    */
FILE *dp ;              /* delay table of the descriptors    */
/*-----------------------------------------------------------------*/

/*******************************************************************
 *                                                                 *
 *                         MAIN PROGRAM                            *
 *                                                                 *
 *******************************************************************/

main(argc, argv)
int argc ;
char *argv[];
 {
  int i, j, k, l, m;
  strcpy(userprg,argv[1]);
  prinpflag = 0 ;
  err_ptr = -1;
  err_count = 0;

/*----------------------PRIMITIVES SUPPORTED---------------------*/
  for (i = 0; i < maxprim; i = i + 1)
   {
    primt[i].normld = 1 ;
    primt[i].fanout = 20 ;
    primt[i].technology = 0 ;
    primt[i].overld = 5 ;
   }
   primsetup(&primt[0]);              /* initialize primitives */
   sys_prims=prim_count;              /* primcount may change, but */
                                      /* we need a copy of its     */
                                      /* starting value            */

/*----------------------KEYWORDS---------------------------------*/
 strcpy(keyword[4], "{") ;
 strcpy(keyword[5], "}") ;
 strcpy(keyword[10],"RISEDEL") ;
 strcpy(keyword[11],"FALLDEL") ;
 strcpy(keyword[12],"TECHNOL") ;
 strcpy(keyword[13],"TTL") ;
 strcpy(keyword[14],"NMOS") ;
 strcpy(keyword[15],"CMOS") ;
 strcpy(keyword[16],"ECL") ;
 strcpy(keyword[17],"FANOUT") ;
 strcpy(keyword[18],"NORMLOA") ;
 strcpy(keyword[19],"OVERLOA") ;
 strcpy(keyword[20],"END") ;
 strcpy(keyword[29],"#");                  /* end of part of edition*/
/*-----------------------------------------------------------------*/

/*-----------------------------------------------------------------*/
 strcpy(key[0],"REPLACE");        /*modify circuit already compiled*/
 strcpy(key[1],"INSERT");         /*insert new gates in the circuit*/
 strcpy(key[2],"DELETE");         /*delete gates in the circuit*/
 strcpy(key[3],"ALTDEL");         /*change delays in the circuit*/
 strcpy(key[4],"ADDPRI");         /*add a printout in the circuit*/
 strcpy(key[5],"DELPRI");         /*delete a printout of the circuit*/
 strcpy(key[6],"ALTINI");         /*change initials of the circuit*/
 strcpy(key[7],"INSOUT");         /*insert output in the circuit*/
 strcpy(key[8],"DELOUT");         /*delete output of the circuit*/
 strcpy(key[9],"ALTGATE");        /*change delays of a gate type*/
 strcpy(key[10],"INSINP");        /*insert inputs in the circuit*/
```

```
 strcpy(key[11],"INSINPG");      /*insert input and gate         */
 strcpy(key[12],"DELINP");       /*delete inputs of the circuit*/
 strcpy(key[13],"DELINPG");      /*delete input and gate */
/*-------------------------------------------------------------------*/

 /* the system will copy all tables that will be used to allow the  */
 /* edition of the circuit.                                         */

  sy = fopen("d:symtable","r");
  fscanf(sy,"]d",&sym_count);
  fscanf(sy,"]d ]d\n",&ord_ini,&ord_pri);
  for (i = 0; i < sym_count; i++)
    {
      fscanf(sy," ]s ]d",symt[i].name,&symt[i].descno);
      fscanf(sy," ]d ]d",&symt[i].funcno,&symt[i].fanld);
      fscanf(sy," ]d ]d",&symt[i].despos,&symt[i].delpos);
      fscanf(sy," ]d ]d",&symt[i].ini_num,&symt[i].pri_num);
      fscanf(sy," ]d\n",&symt[i].pri_val);
    }
  fclose(sy);
  dp = fopen("d:deltable","r");
  fscanf(dp,"]d",&indexl);
  for (i = 0; i < indexl; i++)
    {
      fscanf(dp," ]d ]d",&del_tab[i].indx,&del_tab[i].dsc_nb);
      fscanf(dp," ]d ]d",&del_tab[i].typ_num,&del_tab[i].num_ipt);
      fscanf(dp," ]d ]d\n",&del_tab[i].num_opt,&del_tab[i].val);
    }
  fclose(dp);
  de = fopen("d:descptab","r");
  fscanf(de,"]d\n",&desc_no);
  fscanf(de,"]d\n",&dptr);
  for (i = 0; i < dptr; i++)
    fscanf(de," ]s ]d\n",desct[i].fun,&desct[i].dnum);
  fclose(de);
  nm = fopen ("d:nortable","r") ;
  fscanf(nm," ]d\n",&normcount);
  for (i=0;i<normcount;i++)
       {
         fscanf(nm," ]s ]d\n",nort[i].nom,&nort[i].nmld);
       }
  fclose(nm);
  ip = fopen ("d:inptable","r") ;
  fscanf(ip," ]d\n",&inum);
  for (i=0;i<inum;i++)
    {
        fscanf(ip," ]s ]d",inptab[i].iname,&inptab[i].inp_num);
        fscanf(ip," ]s ]s",inptab[i].inp1,inptab[i].inp2);
        fscanf(ip," ]s ]s",inptab[i].inp3,inptab[i].inp4);
        fscanf(ip," ]s ]s",inptab[i].inp5,inptab[i].inp6);
        fscanf(ip," ]s ]s",inptab[i].inp7,inptab[i].inp8);
        fscanf(ip," ]s ]s",inptab[i].inp9,inptab[i].inp10);
        fscanf(ip," ]d\n",&inptab[i].ifin);
    }
  fclose(ip);

/* The system will fill up all the positions in the input table with */
/* "xxx". This will save time in the program, during the edition.    */

  for (i = inum; i < maxsym; i++)
    {
      strcpy(inptab[i].inp1,"xxx") ;
      strcpy(inptab[i].inp2,"xxx") ;
      strcpy(inptab[i].inp3,"xxx") ;
      strcpy(inptab[i].inp4,"xxx") ;
      strcpy(inptab[i].inp5,"xxx") ;
      strcpy(inptab[i].inp6,"xxx") ;
      strcpy(inptab[i].inp7,"xxx") ;
      strcpy(inptab[i].inp8,"xxx") ;
      strcpy(inptab[i].inp9,"xxx") ;
```

```
    strcpy(inptab[i].inp10,"xxx") ;
  }
sym1 = sym_count - 1 ;
dpt1 = dptr - 1 ;
if (desct[dpt1].dnum < sym1)          /* verification if symtable has */
    val_sym = desct[dpt1].dnum + 1 ;  /* any input that will not be */
else                                  /* used                       */
    val_sym = sym_count ;
rp = fopen(argv[1],"r");
new = 0;
numb_inp = 0;
filecount = 0;
end = 0;
i = index1 - 1;
index = del_tab[i].indx ;
skpc = 0 ;
skpd = 0 ;
for (i = 0; i < val_sym; i++)         /* count how many positions*/
  {                                   /* are occupied in the     */
     skpc = skpc + symt[i].despos ;   /* descriptor file and in the*/
     skpd = skpd + symt[i].delpos ;   /* default delay file       */
  }
while (end != 1)              /* verify if edition was requested */
  {
     getid(rp,33);
     find_key();
     switch(toknn)
        {
            case 0: end = 0;
                    parseid(4);
                    printf("......beginning REPLACE case.\n");
                    repl(0);
                    break;
            case 1: printf("......beginning INSERT case.\n");
                    insgate(0) ;
                    break;
            case 2: printf("......beginning DELETE case.\n");
                    delgate(0) ;
                    break;
            case 3: printf("......beginning DELAY case.\n");
                    end = 0 ;
                    parseid(4);
                    while (1)
                     {
                          /* read the internal*/
                        getid(rp,36);
                        find_token();
                        if (toknn == 5)
                           {
                              end = 1 ;
                              break;
                           }
                        if (toknn == 29)
                           {
                              end = 2 ;
                              break;
                           }
                        strcpy(savbuf,token_buf);
                        findid();
                        target = symt[symid].descno ;
                        primid = symt[symid].funcno ;
                        printf("......making ]s case\n",
                                              symt[symid].name) ;
                        fnddel(0);
                        chgdel();
                     }
                    break;
            case 4: printf("......add PRINTOUT case.\n");
                    val_prt = 0 ;
                    i = ord_pri - 1 ;
```

```
        while (delimiter != 2)
         {
                /* copy the prnt file   */
            filecount = 0 ;
            tp = fopen("d:prnt","r");
            t6 = fopen("d:temp6","w");
            for (k = 1; k < ord_pri; k++)
              {
                 for (l = 0; l < val_sym; l++)
                    {
                       if (symt[l].pri_num == k)
                          {
                             m = symt[l].pri_val ;
                             break ;
                          }
                    }
                 mdy_pri(m,tp,t6);
              }
                    /* read the new printout    */
            getid(rp,33);
            find_token();
            findid() ;
            printf("......making ]s case\n",
                                  symt[symid].name) ;
                   /* insert it in the prnt file  */
            symt[symid].pri_num = ord_pri ;
            for (j = 0; j <= 7 ; j = j + 1 )
               {
                  if (token_buf[j] == '\0')
                     break ;
                  else
                     {
                        fprintf(t6," 28 ]d ]d ",i, j) ;
                        fprintf(t6,"]c ",token_buf[j]);
                        val_prt = val_prt + 1 ;
                        fadvance(4,t6) ;
                     }
               }
             symt[symid].pri_val = val_prt ;
             ord_pri = ord_pri + 1 ;
             val_prt = 0 ;
             fprintf(t6," 29 ]d ]d ",i,
                                  symt[symid].descno) ;
             fadvance(3,t6) ;
             i = i + 1 ;
             chgend(2,tp,t6) ;
             fclose(tp) ;
             fclose(t6) ;
             tp = fopen("d:prnt","w");
             t6 = fopen("d:temp6","r");
             fcopy(t6,tp) ;
             fclose(tp) ;
             fclose(t6) ;
         }
        getid(rp,33);
        find_token();
        if (toknn == 5)
             end = 1 ;
        else
             if (toknn == 29)
                  end = 2 ;
             else
                  error(36) ;
        break;
case 5: printf("......delete PRINTOUT case.\n");
        while (delimiter != 2)
         {
            filecount = 0 ;
            tp = fopen("d:prnt","r");
            t6 = fopen("d:temp6","w");
```

```
                    /* read the printout to delete  */
                getid(rp,33);
                find_token();
                findid() ;
                printf("......making ]s case\n",
                                        symt[symid].name) ;
                target = symt[symid].pri_num ;
                        /*  delete the printout    */
                ersprnt();
                symt[symid].pri_val = 0 ;
                symt[symid].pri_num = 0 ;
                fclose(tp) ;
                fclose(t6) ;
                tp = fopen("d:prnt","w");
                t6 = fopen("d:temp6","r");
                fcopy(t6,tp) ;
                fclose(tp) ;
                fclose(t6) ;
             }
            getid(rp,33);
            find_token();
            if (toknn == 5)
                 end = 1 ;
            else
                 if (toknn == 29)
                      end = 2 ;
                 else
                      error(36) ;
            break;
case 6:     printf("......beginning INITIALIZATION case.\n");
            end = 0 ;
            parseid(4);
            while (1)
             {
                getid(rp,36);
                find_token();
                if (toknn == 5)
                   {
                      end = 1 ;
                      break;
                   }
                if (toknn == 29)
                   {
                      end = 2 ;
                      break;
                   }
                strcpy(savbuf,token_buf);
                findid();
                target = symt[symid].descno ;
                posit = symt[symid].ini_num ;
                printf("......making ]s case\n",
                                        symt[symid].name) ;
                getid(rp,33) ;
                if (delimiter != 2)
                   {
                      error(21) ;
                      break ;
                   }
                 num = atoi(token_buf) ;
                 chginit() ;
             }
            break ;
case 7:     printf("......inserting OUTPUT.\n");
            insgate(1) ;
            getid(rp,33);
            find_token();
            if (toknn == 5)
                 end = 1 ;
            else
                 if (toknn == 29)
```

```
                    end = 2 ;
               else
                    error(36) ;
          break;
 case 8: printf("......deleting OUTPUT.\n");
          delgate(1) ;
          getid(rp,33);
          find_token();
          if (toknn == 5)
               end = 1 ;
          else
               if (toknn == 29)
                    end = 2 ;
               else
                    error(36) ;
          break;
 case 9: printf("......beginning GATE case.\n");
          end = 0 ;
          parseid(4) ;
          while (1)
           {
              getid(rp,36);
              find_token();
              if (toknn == 5)
                 {
                    end = 1 ;
                    break;
                 }
              if (toknn == 29)
                 {
                    end = 2 ;
                    break;
                 }
              strcpy(savbuf,token_buf);
              findprim();
              while (1)
                {
                   fnddel(1) ;
                   for (i = 0; i < val_sym; i++)
                       if (symt[i].funcno == primid)
                          {
                             printf("......making ]s case\n",
                                                 symt[i].name) ;
                             target = symt[i].descno ;
                             chgdel();
                          }
                    if (delimiter == 2)
                        break ;
                }
           }
          break;
case 10: printf("......inserting INPUT.\n");
          chginp();
          end = 0;
          parseid(4);
          repl(1) ;
          break;
case 11: printf("......inserting INPUT and GATE.\n");
          chginp();
          delimiter = 0 ;
          insgate(0);
          break;
case 12: printf("......deleting INPUT.\n");
          ersinp();
          end = 0;
          parseid(4);
          repl(1) ;
          break;
case 13: printf("......deleting INPUT and GATE.\n");
          ersinp();
```

```
                        delimiter = 0 ;
                        delgate(0);
                        break;
                default: error(24) ;
                        break ;
            }
      }
        /* save the modified tables          */
    printf("......Saving tables.\n");
    sy = fopen ("d:symtable","w") ;
    fprintf(sy," ]d\n",sym_count);
    fprintf(sy," ]d ]d\n",ord_ini,ord_pri);
    for (i=0;i<sym_count;i++)
       {
          fprintf(sy," ]s ]d",symt[i].name,symt[i].descno);
          fprintf(sy," ]d ]d",symt[i].funcno,symt[i].fanld);
          fprintf(sy," ]d ]d",symt[i].despos,symt[i].delpos);
          fprintf(sy," ]d ]d",symt[i].ini_num,symt[i].pri_num);
          fprintf(sy," ]d\n",symt[i].pri_val);
       }
    fclose(sy);
    dp = fopen("d:deltable","w");
    fprintf(dp,"]d\n",indexi);
    for (i = 0; i < indexi; i++)
       {
          fprintf(dp," ]d ]d",del_tab[i].indx,del_tab[i].dsc_nb);
          fprintf(dp," ]d ]d",del_tab[i].typ_num,del_tab[i].num_ipt);
          fprintf(dp," ]d ]d\n",del_tab[i].num_opt,del_tab[i].val);
       }
    fclose(dp);
    de = fopen ("d:descptab","w");
    fprintf(de," ]d\n",desc_no);
    fprintf(de," ]d\n",dptr);
    for (i=0;i<dptr;i++)
        fprintf(de," ]s ]d\n",desct[i].fun,desct[i].dnum);
    fclose(de);
    nm = fopen ("d:nortable","w") ;
    fprintf(nm," ]d\n",normcount);
    for (i=0;i<normcount;i++)
       {
          fprintf(nm," ]s ]d\n",nort[i].nom,nort[i].nmld);
       }
    fclose(nm);
    ip = fopen ("d:inptable","w") ;
    fprintf(ip," ]d\n",inum);
    for (i=0;i<inum;i++)
      {
         fprintf(ip," ]s ]d",inptab[i].iname,inptab[i].inp_num);
         fprintf(ip," ]s ]s",inptab[i].inp1,inptab[i].inp2);
         fprintf(ip," ]s ]s",inptab[i].inp3,inptab[i].inp4);
         fprintf(ip," ]s ]s",inptab[i].inp5,inptab[i].inp6);
         fprintf(ip," ]s ]s",inptab[i].inp7,inptab[i].inp8);
         fprintf(ip," ]s ]s",inptab[i].inp9,inptab[i].inp10);
         fprintf(ip," ]d\n",inptab[i].ifin);
      }
    fclose(ip);

    if (err_count != 0)
       error(26) ;              /* Edition discontinued message */
    else
       error(38) ;              /* no errors encountered message    */
    outerror() ;
}
/*-----------------END OF MAIN PROGRAM--------------------------*/

/*****************************************************************
 *                                                               *
 *                      REPL  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
```

```
/* used to make the replacement of one gate by another.          */

repl(code)
int code ;
{
   int i, sub2;
   while (end == 0)
     {
        outpar = -1;
        idout();
        if (end != 0)
           break;
        mdfyfan();              /* update fanload    */
        out_sub = savid[0];  /* save the value of desc being replaced */
        printf("......Replacing ]s\n",symt[out_sub].name) ;
        old_prim = symt[out_sub].funcno;
        old_desc = symt[out_sub].descno;
        for (i = 0; i < dptr; i++)
            if (out_sub == desct[i].dnum)
                break;
                /* positions in the ddf file occupied by the desc */
        skp3 = symt[out_sub].delpos ;
        ant_desc = i;
        skp = 0;
        skp5 = 0 ;
            /* how many spaces are occupied in the dcf and ddf       */
            /* files until the descriptor of the gate to be replaced */
        for (i = 0; i < out_sub; i++)
          {
             skp = skp + symt[i].despos ;
             skp5 = skp5 + symt[i].delpos ;
          }
        filecount = 0 ;
        tc = fopen("d:descf","r");
        t4 = fopen("d:temp4","w");
             /* modify the descriptor in DCF file      */
        cp_sim(skp,tc,t4);
        skp4 = skp ;
        for (i = 0; i < skp2 ; i++)
           {                              /* delete the old descriptor */
              fscanf(tc,"]d",&num1);
           }
        prim(iout, ant_desc);
        new_prim = primid;          /* save the new primitive   */
        idinp(iout);               /* write the new descriptor  */
        skp1 = skpc - skp2 - skp4 ;
        cp_sim(skp1,tc,t4);
        skp = symt[out_sub].despos ;
        skpc = skpc + skp - skp2 ;
        if (code == 0)
           {
              filecount = 0 ;
              td = fopen("d:delf","r");
              t5 = fopen("d:temp5","w");
              tm = fopen("d:modf","r");
              t2 = fopen("d:temp2","w");
              sub2 = ant_desc + 1 ;
                  /* modify the default delays in DDF file */
              cp_sim(skp5,td,t5);
              matgen(ant_desc,sub2) ;
              for (i = 0; i < skp3; i++)
                 {
                    fscanf(td,"]d",&num1);
                 }
              skp4 = skpd - skp5 - skp3 ;
              cp_sim(skp4,td,t5);
              skp1 = symt[out_sub].delpos ;
              skpd = skpd + skp1 - skp3 ;
                  /* delete the modified delays (if have) in SIMDATA */
              if (index1 != 0)
```

```
                {
                    skp = 0 ;
                    filecount = 0 ;
                    dlt_del(old_desc);
                }
            fclose(td);
            fclose(t5);
            fclose(tm);
            fclose(t2);
                    /* restore the used files     */
            td = fopen("d:delf","w");
            t5 = fopen("d:temp5","r");
            tm = fopen("d:modf","w");
            t2 = fopen("d:temp2","r");
            fcopy(t5,td) ;
            fcopy(t2,tm) ;
            fclose(td);
            fclose(t5);
            fclose(tm);
            fclose(t2);
        }
        fclose(tc);
        fclose(t4);
        tc = fopen("d:descf","w");
        t4 = fopen("d:temp4","r");
        fcopy(t4,tc) ;
        fclose(tc);
        fclose(t4);
    }
}
/*--------------------END REPL-----------------------------------*/

/*********************************************************************
 *                                                                   *
 *                      CP_SIM SUBROUTINE                            *
 *                                                                   *
 *********************************************************************/
/* copy the file until the point desired, controlled by the skip */
/* counter                                                       */

cp_sim(code,rx,ry)
int code;
FILE *rx, *ry ;
{
   int i;
   for (i = 0; i < code; i++)
      {
          fscanf(rx,"]d",&num1);
          fprintf(ry,"]d ",num1);
          fadvance(1,ry);
      }
}
/*--------------------END CP_SIM---------------------------------*/

/*********************************************************************
 *                                                                   *
 *                    NEW_SIM SUBROUTINE                             *
 *                                                                   *
 *********************************************************************/
/* copy the INITI and the PRNT files                             */

new_sim()
{
int i;
   filecount = 0 ;
                    /* copy the IPT file              */
   skp = 9 * (ord_ini - 1) ;
   for (i = 0; i < skp; i++)
      {
         fscanf(ti,"]d",&num1);
```

```
            fprintf(t3,"]d ",num1);
            fadvance(1,t3);
        }
    filecount = 0 ;
                            /*  copy the PTT file            */
    copy_sim(tp,t6);
}
/*--------------------END NEW_SIM--------------------------------*/

/*****************************************************************
 *                                                               *
 *                   COPY_SIM SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* copy the PRNT file                                           */

copy_sim(rx,ry)
FILE *rx, *ry ;
{
int i, j, code1;
                            /*   copy the PTT file          */
    for (i = 1; i < ord_pri; i++)
        {
            for (j = 0; j < val_sym; j++)
                {
                  if (symt[j].pri_num == i)
                    {
                      code1 = symt[j].pri_val ;
                      break ;
                    }
                }
            mdy_pri(code1,rx,ry);
        }
    chgend(0,rx,ry) ;
    filecount = 0 ;
}
/*--------------------END COPY_SIM-------------------------------*/

/*****************************************************************
 *                                                               *
 *                     BDREAD SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* This routine reads the block delays for a given function name */

 bdread(s)
 char s[8] ;
 {
  int i, j, num1, x, y, w, z, i1 ;
  FILE *r1 ;
  char p[8] ;
  r1 = fopen("d:bldel","r") ;       /* block delay file */

/*------------------initialize delay matrix to -1 ---------------*/
  for (i = 0; i < maxouts; i = i + 1)
    {
     for (j = 0; j < maxouts; j = j + 1)
       {
        rdmat[i][j] = -1 ;
        fdmat[i][j] = -1 ;
       }
    }
/*-------------------read default block delays-----------------*/
  fscanf(r1,"]s",p) ;
  while (strcmp(p,"END") != 0)
    {
     fscanf(r1,"]d",&num1) ;
     for (i1 = 1; i1 <= num1; i1 = i1 + 1)
       {
        fscanf(r1,"]d ]d ]d ]d",&x, &y, &w, &z) ;
```

```
      if (strcmp(p,s) == 0)
        {
         rdmat[x][y] = w ;
         fdmat[x][y] = z ;
        }
      }
    if (strcmp(p,s) == 0)
      break ;
    fscanf(r1,"]s",p) ;
   }
  fclose(r1) ;
 }
/*--------------------END BDREAD---------------------------------*/

/*****************************************************************
 *                                                               *
 *                     UPFAN  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* update the fanload of the gates                               */

upfan(code)
char code[8];
   {
       int i, j, k ;
       for (j = 0; j < sym_count; j++)
         if (strcmp(code,symt[j].name) == 0)
            break;
       k = symt[j].funcno ;
       for (i = 0; i < normcount; i = i + 1)  /* find name in nort  */
       if (strcmp(nort[i].nom,code) == 0)
         break;
       if (i < normcount)     /* if over ride is used for norm load */
           symt[j].fanld = symt[j].fanld - nort[i].nmld ;
       else                   /* use default value from prim. lib   */
           symt[j].fanld=symt[j].fanld - primt[k].normld;
   }
/*--------------------END UPFAN----------------------------------*/

/*****************************************************************
 *                                                               *
 *                   MDFYFAN SUBROUTINE                          *
 *                                                               *
 *****************************************************************/
/* modify the fanload of the gates                               */

mdfyfan()
  {
     int i;
     for (i = 0; i < inum; i++)
        if ( strcmp(savbuf,inptab[i].iname) == 0)
           break;
     iout = i ;
          /* modify the fanload of the gates that are input   */
          /* of the gate being modified                       */
     upfan(inptab[iout].inp1);
     if (inptab[iout].inp_num > 1)
       {
          upfan(inptab[iout].inp2);
          if (inptab[iout].inp_num > 2)
            {
              upfan(inptab[iout].inp3);
              if (inptab[iout].inp_num > 3)
                {
                   upfan(inptab[iout].inp4);
                   if (inptab[iout].inp_num > 4)
                     {
                        upfan(inptab[iout].inp5);
                        if (inptab[iout].inp_num > 5)
                          {
```

```
                        upfan(inptab[iout].inp6);
                        if (inptab[iout].inp_num > 6)
                          {
                            upfan(inptab[iout].inp7);
                            if (inptab[iout].inp_num > 7)
                              {
                                 upfan(inptab[iout].inp8);
                                 if (inptab[iout].inp_num > 8)
                                   {
                                     upfan(inptab[iout].inp9);
                                     if (inptab[iout].inp_num > 9)
                                       {
                                          upfan(inptab[iout].inp10);
                                       }
                                   }
                              }
                          }
                    }
                }
            }
        }
    }
/*--------------------END UPFAN-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                    DLT_DEL SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* delete the modified delays (if have) from the MODF file       */

dlt_del(code)
int code ;
  {
  int i, j, k, l, m ;
  skp = 0 ;
     /*  verify if the gate appears in the DEL table. If yes      */
     /*  delete the code for the modified delay from the MDF file */
  for (i = 0; i < index1; i++)
     {
        if (del_tab[i].dsc_nb == code)
           {
               cp_sim(skp,tm,t2);
               for (j = 0; j < del_tab[i].val; j++)
                  fscanf(tm," ]d",&num1);
               skp = 0 ;
           }
        else
           skp = skp + del_tab[i].val ;
     }
  l = 0 ;
  i = 0 ;
  while (i < index1)
     {
        j = i ;
        if (del_tab[i].dsc_nb == code)
           {
               while (j < index1)
                 {
                     j = j + 1 ;
                     l = l + 1 ;
                     if (del_tab[j].dsc_nb != code)
                         break;
                 }
                k = i ;
                   /* update deltable                          */
                for (m = j; m < index1; m++)
                  {
                     del_tab[k].dsc_nb = del_tab[m].dsc_nb ;
```

```
                    del_tab[k].typ_num=del_tab[m].typ_num ;
                    del_tab[k].num_ipt=del_tab[m].num_ipt ;
                    del_tab[k].num_opt=del_tab[m].num_opt ;
                    del_tab[k].val = del_tab[m].val ;
                    k = k + 1 ;
                  }
                index1 = index1 - 1 ;
                l = 0 ;
                i = i - 1 ;
             }
          else
             {
                l = 0 ;
             }
       i = i + 1 ;
    }
  cp_sim(skp,tm,t2);
 }
/*--------------------END DLT_DEL-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                    MDY_PRI SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* modify the PRNT file                                          */

mdy_pri(code,rx,ry)
int code;
FILE *rx, *ry ;
{
   int m;
   char z;
   m = 0 ;
   while (m < code)
      {
        fscanf(rx,"]d ]d ]d",&num1,&num2,&num3);
        fprintf(ry,"]d ]d ]d ",num1,num2,num3);
        z = getc(rx) ;
        z = getc(rx) ;
        putc(z,ry);
        fprintf(ry," ");
        fadvance(4,ry);
        m = m + 1 ;
      }
   fscanf(rx,"]d ]d ]d",&num1,&num2,&num3);
   fprintf(ry,"]d ]d ]d ",num1,num2,num3);
   fadvance(3,ry);
}
/*--------------------END MDY_PRI-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                    FNDDEL  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* verify what delay will be modified                            */

 fnddel(code)
 int code ;
 {
                          /* rise or fall delay ?      */
     getid(rp,43);
     find_token();
     savtoken = toknn ;
     temp1 = savtoken ;
     if (delimiter != 6)
        error(29);
                            /* what input ?        */
     getid(rp,33);
```

```
    par1 = atoi(token_buf);
    parm1 = primt[primid].numpar ;
    parm2 = primt[primid].outp ;
    if (par1 > parm1)
       error(39);
    if (delimiter != 1)
       error(31);
                          /* what output ?        */
    getid(rp,33);
    par2 = atoi(token_buf);
    if (par2 > parm2)
       error(40);
                          /* read the delay value   */
    getid(rp,33);
    num = atoi(token_buf);
    if ((code == 0) && (delimiter != 2))
       error(43);
    if ((code == 1) && (delimiter != 1))
       if (delimiter != 2)
           error(43) ;
    savpar1 = par1;
    savpar2 = par2;
 }
/*----------------------END FNDDEL-----------------------------*/

/*****************************************************************
 *                                                               *
 *                   CHGDEL  SUBROUTINE                          *
 *                                                               *
 *****************************************************************/
/* this routine changes the MODF file                           */

 chgdel()
 {
    int i, j, k, m;
    skp = 0 ;
    filecount = 0 ;
    tm = fopen("d:modf","r");
    t2 = fopen("d:temp2","w");
    if (num == 0)                 /* return to the default values: */
      {                           /* delete the code from MDF file */
        for (i = 0; i < index1; i ++)
           {                      /* and update DEL table          */
              if ((del_tab[i].dsc_nb == target) &&
                  (del_tab[i].typ_num == temp1) &&
                  (del_tab[i].num_ipt == savpar1) &&
                  (del_tab[i].num_opt == savpar2))
                  {
                       cp_sim(skp,tm,t2);
                       for (j = 0; j < del_tab[i].val;j++)
                           fscanf(tm,"]d",&num1);
                       skp = 0;
                       m = i;
                       j = i + 1 ;
                       for (k = j; k < index1; k++)
                           {
                              del_tab[m].dsc_nb = del_tab[k].dsc_nb ;
                              del_tab[m].typ_num = del_tab[k].typ_num ;
                              del_tab[m].num_ipt = del_tab[k].num_ipt ;
                              del_tab[m].num_opt = del_tab[k].num_opt ;
                              del_tab[m].val = del_tab[k].val ;
                              m = m + 1 ;
                           }
                        i = i - 1;
                        index1 = index1 - 1;
                        index = index - 1;
                  }
              else
                  {
                        skp = skp + del_tab[i].val;
```

```
                    }
                }
            cp_sim(skp,tm,t2);
            endf = 1 ;
        }
    else                /*  modify delays already modified   */
        {
            for (i = 0; i < index1; i++)
                {
                    if ((del_tab[i].dsc_nb == target) &&
                        (del_tab[i].typ_num == temp1) &&
                        (del_tab[i].num_ipt == savpar1) &&
                        (del_tab[i].num_opt == savpar2))
                        {
                            cp_sim(skp,tm,t2);
                            j = del_tab[i].val - 1 ;
                            for (k = 0; k < j; k ++)
                              {
                                 fscanf(tm,"]d ",&num1);
                                 fprintf(t2,"]d ",num1);
                                 fadvance(1,t2);
                              }
                            fprintf(t2,"]d ",num);
                            fscanf(tm,"]d ",&num1);
                            fadvance(1,t2);
                            endf = 1;
                            skp = 0;
                        }
                    else
                        {
                            skp = skp + del_tab[i].val ;
                        }
                }
            cp_sim(skp,tm,t2);
        }
    skp = 0 ;
    if (endf == 0)                /* insert new delays         */
        {
            if (temp1 == 10)
                {
                    rfdel(0,par1,par2,t2);
                }
            else
                {
                    rfdel(1,par1,par2,t2);
                }
        }
    fclose(t2);
    fclose(tm);
    endf = 0 ;
    t2 = fopen("d:temp2","r");
    tm = fopen("d:modf","w");
    fcopy(t2,tm);
    fclose(t2) ;
    fclose(tm) ;
}
/*----------------------END CHGDEL-----------------------------*/

/*****************************************************************
 *                                                               *
 *                   CHGINIT  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* modify the initialization of the internals                    */

chginit()
{
    int i ;
    skp = 0 ;
    filecount = 0 ;
```

```
ti = fopen("d:initi","r");
t3 = fopen("d:temp3","w");
if (ord_ini == 1)              /* no previous initialization     */
   {
     if (num != 3)             /* insert new initial value       */
       {
          fprintf(t3,"20 21 23 24 25 26 ") ;
          fadvance(6,t3) ;
          fprintf(t3,"]d 27 ]d ",target,num) ;
          fadvance(2,t3) ;
          symt[symid].ini_num = 1 ;
          ord_ini = 2 ;
       }
     else
       {
          fcopy(ti,t3) ;
       }
   }
else
   {
      if (num == 3)            /* delete initialization   */
        {
           if (posit != 0)
              {
                 skp = (posit - 1) * 9 ;
                 skp1 = (ord_ini - posit - 1) * 9 ;
                 cp_sim(skp,ti,t3) ;
                 for (i = 0; i < 9; i++)
                     fscanf(ti,"]d",&num1);
                 if (posit == 1)
                     {
                        fprintf(t3,"20 21 ");
                        fscanf(ti,"]d ]d",&num1,&num2) ;
                        fadvance(2,t3);
                        for (i = 0; i < 7; i++)
                           {
                              fscanf(ti,"]d",&num1) ;
                              fprintf(t3,"]d ",num1);
                              fadvance(1,t3) ;
                           }
                        skp1 = skp1 - 9 ;
                     }
                 cp_sim(skp1,ti,t3);
                 symt[symid].ini_num = 0 ;
                 for (i = 0; i < val_sym; i++)
                     if(symt[i].ini_num > posit)
                         symt[i].ini_num = symt[i].ini_num - 1 ;
                 ord_ini = ord_ini - 1 ;
              }
           else
              {
                 fcopy(ti,t3) ;
              }
        }
      else
        {                 /* if is not the first descriptor  */
           if (posit != 0)     /* in the file                   */
              {
                 skp = (posit - 1) * 9 ;
                 skp1 = (ord_ini - posit - 1) * 9 ;
                 cp_sim(skp,ti,t3) ;
                 for (i = 0; i < 8; i++)
                    {
                       fscanf(ti,"]d",&num1);
                       fprintf(t3,"]d ",num1);
                       fadvance(1,t3) ;
                    }
                 fscanf(ti,"]d",&num1);
                 fprintf(t3,"]d ",num);
                 fadvance(1,t3) ;
```

```
                    cp_sim(skp1,ti,t3);
                }
            else
                {
                    skp = (ord_ini - 1) * 9 ;
                    cp_sim(skp,ti,t3) ;
                    fprintf(t3,"20 22 23 24 25 26 ");
                    fadvance(6,t3) ;
                    fprintf(t3,"]d 27 ]d ",target,num);
                    fadvance(3,t3);
                    symt[symid].ini_num = ord_ini ;
                    ord_ini = ord_ini + 1 ;
                }
        }
    }
    fclose(t3);
    fclose(ti);
    ti = fopen("d:initi","w");
    t3 = fopen("d:temp3","r");
    fcopy(t3,ti);
    fclose(t3) ;
    fclose(ti) ;
}
/*----------------------END CHGINIT----------------------------*/

/*****************************************************************
 *                                                               *
 *                    CHGEND  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* this routine modifies the PRNT file                          */

chgend(code,rx,ry)
int code ;
FILE *rx, *ry ;
{
    fscanf(rx,"]d ]d",&num1,&num2);
    fprintf(ry,"]d ]d ",num1,num2);
    fscanf(rx,"]d",&num1);
    if (code == 1)              /* delete printout   */
        num1 = num1 - 1 ;
    if (code == 2)              /* insert printout   */
        num1 = num1 + 1 ;
    fprintf(ry,"]d ",num1);
    fscanf(rx,"]d ]d",&num1,&num2);
    fprintf(ry,"]d ]d ",num1,num2);
    fadvance(5,ry);
    fprintf(ry,"\n");
}
/*----------------------END CHGEND----------------------------*/

/*****************************************************************
 *                                                               *
 *                    ERSPRNT SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* this routine erases a printout from the circuit              */

ersprnt()
{
int k, l, m, n ;
  if (target != 0)
    {
       for (k = 1; k < ord_pri; k++)
         {
            for (l = 0; l < val_sym; l++)
              {
                 if (symt[l].pri_num == k)
                   {
                      m = symt[l].pri_val ;
```

```
                    break ;
                }
            }
        if (k < target)
            mdy_pri(m,tp,t6);
        if (k == target)
            {
                n = 0 ;
                while (n < m)
                    {
                        fscanf(tp,"]d ]d ]d",&num1,&num2,&num3) ;
                        z = getc(tp) ;
                        z = getc(tp) ;
                        n = n + 1 ;
                    }
                fscanf(tp,"]d ]d ]d",&num1,&num2,&num3) ;
            }
        if (k > target)
            {
                n = 0 ;
                while (n < m)
                    {
                        fscanf(tp,"]d",&num1);
                        fprintf(t6,"]d ",num1);
                        fscanf(tp,"]d",&num2);
                        num2 = num2 - 1 ;
                        fprintf(t6,"]d ",num2);
                        fscanf(tp,"]d",&num3);
                        fprintf(t6,"]d ",num3);
                        z = getc(tp) ;
                        z = getc(tp) ;
                        putc(z,t6);
                        fprintf(t6," ");
                        fadvance(4,t6);
                        n = n + 1 ;
                    }
                fscanf(tp,"]d",&num1);
                fprintf(t6,"]d ",num1);
                fscanf(tp,"]d",&num2);
                num2 = num2 - 1 ;
                fprintf(t6,"]d ",num2);
                fscanf(tp,"]d",&num3);
                fprintf(t6,"]d ",num3);
                fadvance(3,t6);
                symt[l].pri_num = symt[l].pri_num - 1 ;
            }
        }
    ord_pri = ord_pri - 1 ;
                /* copy the end of PRNT          */
    chgend(1,tp,t6) ;
    }
  else
    {
    fcopy(tp,t6) ;
    }
}
/*----------------------END ERSPRNT----------------------------*/

/*****************************************************************
 *                                                               *
 *                    INSGATE SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* this routine insers a gate in the circuit                     */

insgate(code)
int code ;
{
int i, j ;
            /* save the original values of tables */
```

```
old_ptr = dptr;
old_sym = sym_count;
old_val = val_sym;
old_ipt = inum ;
old_del = index1 ;
desc_old = desc_no ;
           /* verify if has more than one insertion     */
while (delimiter != 2)
   {
       parseid(maxkey);
               /*  update tables                        */
       if (val_sym == sym_count)
          {
               update(-2,sym_count);
               desc_no = desc_no + 1 ;
               sym_count = sym_count + 1 ;
               val_sym = sym_count ;
          }
       else
          {
               for (i = sym_count; i > val_sym; i--)
                  {
                      j = i - 1 ;
                      strcpy(symt[i].name,symt[j].name);
                      symt[i].descno = symt[j].descno ;
                      symt[i].funcno = symt[j].funcno ;
                      symt[i].fanld = symt[j].fanld ;
                      symt[i].despos=symt[j].despos ;
                      symt[i].delpos=symt[j].delpos ;
                      symt[i].ini_num=symt[j].ini_num ;
                      symt[i].pri_num=symt[j].pri_num ;
                      symt[i].pri_val=symt[j].pri_val ;
                  }
               update(-2,val_sym);
               val_sym = val_sym + 1 ;
               sym_count = sym_count + 1 ;
               desc_no = desc_no + 1 ;
          }
       new = new + 1 ;
   }
end = 0;
parseid(4);
          /* make the insertion in the tables and in the files    */
while (end == 0)
   {
      while (new != 0)
          {
              filecount = 0 ;
              tc = fopen("d:descf","r");
              t4 = fopen("d:temp4","w");
              td = fopen("d:delf","r");
              t5 = fopen("d:temp5","w");
              idout();
              if (end != 0)
                 {
                     error(0) ;
                     break;
                 }
              strcpy(inptab[inum].iname, savbuf);
              printf("......Inserting ]s\n",savbuf) ;
              cp_sim(skpc,tc,t4);
                              /* insert new descriptor in DESCF*/
              prim(inum, dptr);
              idinp(inum);
              skpc = skpc + symt[valact].despos ;
              new = new - no_new ;
              inptab[inum].ifin = 1;
              inum = inum + 1;
              filecount = 0 ;
              cp_sim(skpd,td,t5);
```

```
                        /* insert default delays of new desc*/
            matgen(old_ptr,dptr);
            skp2 = symt[valact].delpos ;
            skpd = skpd + skp2 ;
            fclose(tc);
            fclose(t4);
            fclose(td);
            fclose(t5);
            old_sym = old_sym + no_new ;
            old_val = old_val + no_new ;
            old_ptr = old_ptr + no_new ;
                        /*  restore the files      */
            tc = fopen("d:descf","w");
            t4 = fopen("d:temp4","r");
            td = fopen("d:delf","w");
            t5 = fopen("d:temp5","r");
            fcopy(t4,tc) ;
            fcopy(t5,td) ;
            fclose(tc);
            fclose(t4);
            fclose(td);
            fclose(t5);
        }
      if (code == 0)
        {
          if (end != 0)
            {
                error(1) ;
                break;
            }
          repl(1);
        }
      else
        {
          end = 1 ;
        }
   }
}
/*-----------------------END INSGATE-----------------------------*/

/*****************************************************************
 *                                                               *
 *                    CHGINP  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* this routine inserts an input in the circuit                  */

chginp()
{
int i, j ;
              /* verify if has more than one insertion     */
while (delimiter != 2)
  {
      filecount = 0 ;
      tc = fopen("d:descf","r");
      t4 = fopen("d:temp4","w");
      parseid(maxkey);
      printf("......inserting input ]s\n",token_buf);
                        /*  update tables                 */
      for (i = sym_count; i > 0; i--)
        {
            j = i - 1 ;
            strcpy(symt[i].name,symt[j].name);
            symt[i].descno = symt[j].descno ;
            symt[i].funcno = symt[j].funcno ;
            symt[i].fanld = symt[j].fanld ;
            symt[i].despos=symt[j].despos ;
            symt[i].delpos=symt[j].delpos ;
            symt[i].ini_num=symt[j].ini_num ;
            symt[i].pri_num=symt[j].pri_num ;
```

```
            symt[i].pri_val=symt[j].pri_val ;
         }
             /* put the values for the new input on the tables  */
      strcpy(symt[0].name,token_buf);
      symt[0].descno = desc_no ;
      symt[0].funcno = 0 ;
      symt[0].fanld = 0 ;
      symt[0].despos = 11 ;
      symt[0].delpos = 0 ;
      symt[0].ini_num = 0 ;
      symt[0].pri_num = 0 ;
      symt[0].pri_val = 0 ;
      for (i = 0; i < dptr; i++)
         desct[i].dnum = desct[i].dnum + 1 ;
           /* put the code for the new input in the DCF file  */
      code_input(desc_no) ;
      cp_sim(skpc,tc,t4) ;
      skpc = skpc + 11 ;
      val_sym = val_sym + 1 ;
      sym_count = sym_count + 1 ;
      desc_no = desc_no + 1 ;
      fclose(tc);
      fclose(t4);
      tc = fopen("d:descf","w");
      t4 = fopen("d:temp4","r");
      fcopy(t4,tc) ;
      fclose(tc);
      fclose(t4);
   }
 }
/*----------------------END CHGINP-----------------------------*/

/*********************************************************************
 *                                                                   *
 *                    DELGATE SUBROUTINE                             *
 *                                                                   *
 *********************************************************************/
/* this routine erases a gate from the circuit                       */

 delgate(code)
 int code ;
 {
 int i, j ;
                    /* until all deletions be done      */
 while (delimiter != 2)
    {
       getid (rp,33);
       find_token();
                     /* save the values of the gates */
                     /* that are being deleted from  */
                     /* the tables                   */
       for (i = 0; i < val_sym; i++)
          if (strcmp(symt[i].name,token_buf) == 0)
             break;
       del_sym = i;
       old_func = symt[del_sym].funcno ;
       strcpy(savbuf,token_buf);
       for (i = 0; i < inum; i++)
          if (strcmp(inptab[i].iname,token_buf) == 0)
             break;
       del_ipt = i;
       for (i = 0; i < dptr; i++)
          if (desct[i].dnum == del_sym)
             break;
       del_dct = i;
       mdfyfan();
       tc = fopen("d:descf","r");
       t4 = fopen("d:temp4","w");
       td = fopen("d:delf","r");
       t5 = fopen("d:temp5","w");
```

```
tp = fopen("d:prnt","r");
t6 = fopen("d:temp6","w");
tm = fopen("d:modf","r");
t2 = fopen("d:temp2","w");
printf("......Deleting ]s\n", symt[del_sym].name) ;
skp = 0;
skp5 = 0 ;
for (i = 0; i < del_sym; i++)
  {
     skp = skp + symt[i].despos ;
     skp5 = skp5 + symt[i].delpos ;
  }
              /* delete the descriptor from DESCF     */
filecount = 0 ;
cp_sim(skp,tc,t4);
tot_val = symt[del_sym].despos;
del_val = symt[del_sym].delpos;
for (i = 0; i < tot_val; i++)
   fscanf(tc,"]d",&num1);
skp1 = skpc - tot_val - skp ;
skpc = skpc - tot_val ;
skp2 = del_val ;
cp_sim(skp1,tc,t4);
              /* delete the default delays of deleted */
              /* gate from DELF                       */
filecount = 0 ;
cp_sim(skp5,td,t5);
for (i = 0; i < skp2; i++)
   fscanf(td,"]d",&num1);
skp1 = skpd - skp2 - skp5 ;
skpd = skpd -skp2 ;
cp_sim(skp1,td,t5) ;
               /* delete the modified delays of the   */
               /* desc. (if have) from MODF and       */
               /* update deltable                     */
filecount = 0 ;
if (index1 != 0)
   {
      target = symt[del_sym].descno ;
      dlt_del(target);
   }
              /* delete the initialization values of */
              /* desc. (if have) from INITI and      */
              /* update symtable                     */
if (ord_ini != 1)
   {
       posit = symt[del_sym].ini_num ;
       filecount = 0 ;
       num = 3 ;
       chginit() ;
   }
              /* delete the printout of the desc.    */
              /* (if have) from PRNT and update      */
              /* symtable                            */
filecount = 0 ;
target = symt[del_sym].pri_num ;
ersprnt() ;
fclose(tp);
fclose(t6);
fclose(tc);
fclose(t4);
fclose(td);
fclose(t5);
fclose(tm);
fclose(t2);
              /* restore the files                   */
tc = fopen("d:descf","w");
t4 = fopen("d:temp4","r");
td = fopen("d:delf","w");
t5 = fopen("d:temp5","r");
```

```
        tp = fopen("d:prnt","w");
        t6 = fopen("d:temp6","r");
        tm = fopen("d:modf","w");
        t2 = fopen("d:temp2","r");
        fcopy(t4,tc) ;
        fcopy(t5,td) ;
        fcopy(t6,tp) ;
        fcopy(t2,tm) ;
        fclose(tc);
        fclose(t4);
        fclose(tm);
        fclose(t2);
        fclose(td);
        fclose(t5);
        fclose(tp);
        fclose(t6);
                          /* update all tables          */
        olddel_sym = del_sym ;
        del_sym = del_sym + 1;
        for (i = del_sym; i < sym_count; i++)
           {
               j = i - 1;
               strcpy(symt[j].name,symt[i].name);
               symt[j].descno = symt[i].descno;
               symt[j].funcno = symt[i].funcno;
               symt[j].fanld = symt[i].fanld;
               symt[j].despos = symt[i].despos;
               symt[j].delpos = symt[i].delpos;
               symt[j].ini_num = symt[i].ini_num;
               symt[j].pri_num = symt[i].pri_num;
               symt[j].pri_val = symt[i].pri_val;
           }
        sym_count = sym_count - 1;
        del_dct = del_dct + 1;
        for (i = del_dct; i < dptr; i++)
           {
               j = i - 1;
               strcpy(desct[j].fun,desct[i].fun);
               if (desct[i].dnum > olddel_sym)
                  desct[j].dnum = desct[i].dnum - 1 ;
               else
                  desct[j].dnum = desct[i].dnum ;
           }
        dptr = dptr - 1;
        del_ipt = del_ipt + 1;
        if (del_ipt <= inum)
           {
              for (i = del_ipt; i < inum; i++)
                 {
                     j = i - 1;
                     strcpy(inptab[j].iname,inptab[i].iname) ;
                     inptab[j].inp_num = inptab[i].inp_num ;
                     strcpy(inptab[j].inp1,inptab[i].inp1) ;
                     strcpy(inptab[j].inp2,inptab[i].inp2) ;
                     strcpy(inptab[j].inp3,inptab[i].inp3) ;
                     strcpy(inptab[j].inp4,inptab[i].inp4) ;
                     strcpy(inptab[j].inp5,inptab[i].inp5) ;
                     strcpy(inptab[j].inp6,inptab[i].inp6) ;
                     strcpy(inptab[j].inp7,inptab[i].inp7) ;
                     strcpy(inptab[j].inp8,inptab[i].inp8) ;
                     strcpy(inptab[j].inp9,inptab[i].inp9) ;
                     strcpy(inptab[j].inp10,inptab[i].inp10) ;
                     inptab[j].ifin = inptab[i].ifin;
                 }
              inum = inum - 1;
           }
        val_sym = val_sym - 1 ;
     }
  if (code == 0)
     {
```

```
      if (end == 0)
         {
            parseid(4);
            repl(1);
         }
      else
            error(1);
   }
}
/*----------------------END DELGATE-----------------------------*/

/*****************************************************************
 *                                                               *
 *                    ERSINP  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* this routine erases an input from the circuit                 */

ersinp()
{
int i, j ;
                /* verify if has more than one deletion     */
while (delimiter != 2)
   {
      filecount = 0 ;
      tc = fopen("d:descf","r");
      t4 = fopen("d:temp4","w");
      parseid(maxkey);
      printf("......deleting input ]s\n",token_buf);
                        /*  update tables                   */
      for (i = 0; i < val_sym; i++)
         if (strcmp(symt[i].name,token_buf) == 0)
            break;
      del_sym = i;
      skp = 0;
      for (i = 0; i < del_sym; i++)
           skp = skp + symt[i].despos ;
                  /* delete the descriptor from DESCF     */
      cp_sim(skp,tc,t4);
      tot_val = symt[del_sym].despos;
      for (i = 0; i < tot_val; i++)
         fscanf(tc,"]d",&num1);
      skp1 = skpc - tot_val - skp ;
      skpc = skpc - tot_val ;
      cp_sim(skp1,tc,t4) ;
      del_sym = del_sym + 1 ;
                        /*  update the tables    */
      for (i = del_sym; i < sym_count; i++)
         {
             j = i - 1 ;
             strcpy(symt[j].name,symt[i].name);
             symt[j].descno = symt[i].descno ;
             symt[j].funcno = symt[i].funcno ;
             symt[j].fanld = symt[i].fanld ;
             symt[j].despos=symt[i].despos ;
             symt[j].delpos=symt[i].delpos ;
             symt[j].ini_num=symt[i].ini_num ;
             symt[j].pri_num=symt[i].pri_num ;
             symt[j].pri_val=symt[i].pri_val ;
         }
      for (i = 0; i < dptr; i++)
         desct[i].dnum = desct[i].dnum - 1 ;
      val_sym = val_sym - 1 ;
      sym_count = sym_count - 1 ;
      fclose(tc);
      fclose(t4);
      tc = fopen("d:descf","w");
      t4 = fopen("d:temp4","r");
      fcopy(t4,tc) ;
      fclose(tc);
```

```
        fclose(t4);
    }
}
/*-----------------------END ERSINP-------------------------------*/
```

APPENDIX E
THE PRECOMP PROGRAM FOR THE EDITOR

```
/*****************************************************************
 *                                                               *
 *                    Precomp   Program                          *
 *                          for the                              *
 *                          Editor                               *
 *                                                               *
 *                  VERSION 3.1, 14 Apr 1987.                    *
 *     Original version developed under MSDOS and PCDOS by       *
 *     Julio Cesar Lopes de Albuquerque at Naval Postgraduate    *
 *     School. Use with EDITOR version 3.1.                      *
 *                                                               *
 *****************************************************************/

#include "\lc\stdio.h"      /*<stdio.h> in DOS 3.x environment   */

#define maxkey 30             /* maximum number of keywords      */
#define maxsym 500
#define maxprim 100           /* maximum number of primitives    */
#define maxouts 32            /* maximum of 32 outputs per prim. */
#define maxnorm 50
#define maxk 14               /* user options for the program    */

/*------------------GLOBAL DECLARATIONS--------------------------*/
/*---------------------------------------------------------------*/

/*----------------DATA STRUCTURES--------------------------------*/
          /* the names and meanings are the same from the EDITOR */
struct sym_tab {
  char name[8] ;
  int descno, funcno ;
  int fanld;
  int despos, delpos;
  int ini_num, pri_num;
  int pri_val;
 } ;

struct desc_tab {
  char fun[8] ;
  int dnum ;
 } ;

struct norm_tab {
  char nom[8] ;
  int nmld ;
 } ;

struct prim_tab {
  char nam[8] ;
  char nam2[8] ;
  int numpar, outp ;
  int normld, fanout ;
  int technology, overld ;
 } ;

struct err_stack {
  char nm[8] ;
  int errno ;
 } ;

struct exp_tab {
  char fname[8] ;
```

```
  int fnum ;
 } ;

struct namepair {
  char e_from[8];
  char e_to[8];
 };

struct swapname {
  char sname[8];
  int used;
  };

struct inp_name {
  char iname[8];
  int inp_num;
  char inp1[8], inp2[8];
  char inp3[8], inp4[8];
  char inp5[8], inp6[8];
  char inp7[8], inp8[8];
  char inp9[8], inp10[8];
  int ifin;
  };

struct tab_del {
  int indx, dsc_nb, typ_num;
  int num_ipt, num_opt, val;
  };
/*-----------------------------------------------------------------*/

/*---------------STORAGE ALLOCATION-------------------------------*/
extern struct sym_tab symt[maxsym] ;
extern struct desc_tab desct[maxsym] ;
extern struct norm_tab nort[maxnorm] ;
extern struct prim_tab primt[maxprim], *primptr ;
extern struct err_stack errt[5] ;  /* max of 5 errors per line      */
extern struct inp_name inptab[maxsym];
extern struct tab_del del_tab[100];

extern int err_ptr ;       /* error table pointer (count)         */
                           /* for one line.                       */
extern int desc_no , sym_count, symid ;   /* desc_no = desc. count */
extern int dptr, descid, lim ;    /* descriptor table indices      */
                           /* dptr = descriptor table cnt         */
extern int end, outpar;
extern int savid[maxouts] ;
extern int savprim, no_new;

extern int savn[maxouts];
extern int matcount ;         /* delay matrix count                */
extern int delimiter, bb ;    /* delimiter = delimiter type        */
                           /* bb = buff[80] index (line)          */
extern int rdmat[maxouts][maxouts], fdmat[maxouts][maxouts] ;
                           /* rise and fall delay matrices        */
extern int toknn, err_count ;     /* err_count  = error count      */
extern int normcount ;            /* normtable count               */
extern int filecount;             /* poutcount used for debugging  */
extern int prinpflag;             /* controls printing of source   */
extern int prim_count, primid ;   /* prim_count = # of primitives  */
extern char token_buf[8], savbuf[8], buff[80] ; /* buff = 1 line   */
extern char keyword[maxkey][8] ;  /* keyword table                 */
extern char key[maxk][8];         /* key table                     */
extern int hashtable[100];        /* the hash table                */
extern int hashcount;             /* number of items in hashtable  */
extern int new;
extern int lim, index, index1 ;
extern int parm2,parm1, target ;
extern int par1, par2, savpar1 ;
extern int savpar2, num ;
extern int valact, skp2 ;
```

```
extern int savn[maxouts];

extern FILE *r1 ;            /* pointer to input data file        */
extern FILE *t1 ;            /* pointer to temp file              */
extern FILE *t2 ;
extern FILE *t3 ;
extern FILE *t4 ;
extern FILE *t5 ;
extern FILE *t6 ;
extern FILE *tc ;
extern FILE *td ;
extern FILE *tm ;
extern FILE *ti ;
extern FILE *tp ;
extern FILE *rp ;            /* read pointer to input data file   */
extern FILE *sy ;            /* symt table stored for edition     */
extern FILE *de ;            /* desct table stored for edition    */
extern FILE *nm ;            /* nort table stored for edition     */
extern FILE *ip ;            /* input table stored for edition    */
extern FILE *dp ;            /* delay table of the descriptors    */
/*-------------------------------------------------------------*/

/*************************************************************
 *                                                           *
 *                    IDOUT  SUBROUTINE                      *
 *                                                           *
 *************************************************************/
/* This subroutine will verify what output or internal variable  */
/* will be used in the present case.                             */

 IDOUT()
 {
    outpar = -1;
    while (1)
     {
      getid(rp,46);
      find_token();
      strcpy(savbuf, token_buf);
      if (toknn == 5)     /*  if } found, end edition           */
       {
        end = 1 ;
        break ;
       }
      if (toknn == 29)    /*  if # found, end this part         */
       {
        end = 2 ;
        break ;
       }
      if (toknn < maxkey) /* left hand side should not be a keyw.*/
        error(25) ;
      outpar = outpar + 1 ;
      findprim() ;
      if (primid < prim_count)
        error(25) ;       /* output name is a keyword          */
      findid() ;          /* find the symbol table index       */
      if (symid >= 0)
       {                       /* save output indices in an array */
        savid[outpar] = symid ;
        savn[outpar] = symt[symid].descno ;
       }
      valact = savid[0] ;
      skp2 = symt[valact].despos ;
      symt[valact].despos = 0 ;
      if (delimiter == 4) /* '=' should follow the output names  */
        break ;
      else
       {
        if (delimiter != 1)
          error(31) ;     /* ',' expected after each name      */
       }
```

```
    }
  }
/*----------------------END IDOUT-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                      IDINP SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* This subroutine will make the scan of all the inputs that will*/
/* be used in a determined gate.                                 */

 IDINP(code)
 int code;
 {
    int savpar, savinp ;
    int fwdp ;
    fwdp = 0 ;
    savpar = primt[primid].numpar ;  /* number of inputs          */
    no_new = primt[primid].outp ;    /* number of outputs         */
    savinp = 0 ;
    savprim = primid ;               /* save primitive            */
    strcpy(savbuf, token_buf) ;  /* save function name/type       */
    while (savpar != 0)    /* while all inputs have been scanned */
     {
      parseid(maxkey) ;    /* parameter of function             */
      switch(savinp)       /* update  inptable                  */
         {
             case 0 : strcpy(inptab[code].inp1,token_buf);
                      break;
             case 1 : strcpy(inptab[code].inp2,token_buf);
                      break;
             case 2 : strcpy(inptab[code].inp3,token_buf);
                      break;
             case 3 : strcpy(inptab[code].inp4,token_buf);
                      break;
             case 4 : strcpy(inptab[code].inp5,token_buf);
                      break;
             case 5 : strcpy(inptab[code].inp6,token_buf);
                      break;
             case 6 : strcpy(inptab[code].inp7,token_buf);
                      break;
             case 7 : strcpy(inptab[code].inp8,token_buf);
                      break;
             case 8 : strcpy(inptab[code].inp9,token_buf);
                      break;
             case 9 : strcpy(inptab[code].inp10,token_buf);
                      break;
         }
      savinp = savinp + 1 ;
      findid() ;           /* find parameter's location in the  */

      connect(0,savn,fwdp);/* generate code and update fanld    */
      if (savpar == 1)
       {
        if (delimiter != 5)  /* ')' expected after the last arg. */
          error(32) ;
       }
      savpar = savpar - 1 ;
      if (savpar != 0)
       {
        if (delimiter != 1)  /*, expected after first parameter  */
          error(31) ;
        parseid(maxkey) ;    /*  get next argument               */
        switch(savinp)       /* update  inptable                 */
           {
             case 0 : strcpy(inptab[code].inp1,token_buf);
                      break;
             case 1 : strcpy(inptab[code].inp2,token_buf);
                      break;
```

```
            case 2 : strcpy(inptab[code].inp3,token_buf);
                     break;
            case 3 : strcpy(inptab[code].inp4,token_buf);
                     break;
            case 4 : strcpy(inptab[code].inp5,token_buf);
                     break;
            case 5 : strcpy(inptab[code].inp6,token_buf);
                     break;
            case 6 : strcpy(inptab[code].inp7,token_buf);
                     break;
            case 7 : strcpy(inptab[code].inp8,token_buf);
                     break;
            case 8 : strcpy(inptab[code].inp9,token_buf);
                     break;
            case 9 : strcpy(inptab[code].inp10,token_buf);
                     break;
          }
      savinp = savinp + 1 ;
        if (savpar > 1)
         {
          if (delimiter != 1)/* , expected after argument        */
            error(31) ;
         }
        else
         {
          if (delimiter != 5)
            error(32) ;       /* ) expected after last arg.      */
         }
        findid() ;            /* find symbol table index for the */
                              /* input name                      */
        connect(1,savn,fwdp); /* generate code and update fanld  */
        savpar = savpar - 1 ;
        fwdp = fwdp + 1 ;
        if (outpar > 0)             /* multioutput case */
          outpar = outpar - 1 ;
        else
         {
          if (savpar != 0)          /* multiinput case  */
           {
            fprintf(t4,"1 ]d 15 ]d ", savn[fwdp - 1], desc_no) ;
            fprintf(t4,"1 ]d 16 ]d ", desc_no, savn[fwdp - 1]) ;
            fadvance(8,t4) ;
            symt[valact].despos = symt[valact].despos + 8 ;
            savn[fwdp] = desc_no ;
            desc_no = desc_no + 1 ;
           }
         }
      } /* end if                          */
    }
 }
/*---------------------END IDINP-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                  FIND_ID  SUBROUTINE                          *
 *                                                               *
 *****************************************************************/
/* finds the symbol table index. An error message is generated if*/
/* the name does not exist */

findid()
 {
  int i ;
  for (i = 0; i < sym_count; i = i + 1)
    if ( strcmp(token_buf,symt[i].name) == 0)
      break ;
  if (i == sym_count)       /* name not found in the symbol table */
   {
    error(28) ;             /* undeclared name                    */
    symid = -1 ;
```

```
    }
  else
    symid = i ;
 }
/*--------------------END FIND_ID--------------------------------*/

/*****************************************************************
 *                                                               *
 *                  FIND_PRIM SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* finds the primitive that will be used in the PRIMITIV.DAT file*/

findprim()
 {
  int i ;
  for (i = 0; i < prim_count; i = i + 1)
    if ( strcmp(token_buf,primt[i].nam2) == 0)
      break ;
  primid = i ;
 }
/*--------------------END FIND_PRIM------------------------------*/

/*****************************************************************
 *                                                               *
 *                  OUTERROR SUBROUTINE                          *
 *                                                               *
 *****************************************************************/
/* This routine outputs all errors encountered in a line after   */
/* entire line has been read.                                    */

 outerror()
  {
   int i;
   i = 0 ;
   while (err_ptr >= 0)       /* if error count for a line is > 0 */
    {                         /* print all errors encountered     */
     errmessage(errt[i].errno) ;
     i = i + 1 ;
     err_ptr = err_ptr - 1 ;
    }
  }
/*--------------------END OUTERROR-------------------------------*/

/*****************************************************************
 *                                                               *
 *                     ERROR SUBROUTINE                          *
 *                                                               *
 *****************************************************************/
/* This routine enters the error number and the name of the wrong*/
/* identifier in the errt (error table). The errors are printed  */
/* after the whole line has been scanned.                        */

 error(i)
  int i ;       /* i = error number */
  {
   err_ptr = err_ptr + 1 ;          /* error count for one line  */
   errt[err_ptr].errno = i ;        /* errno = error number      */
   strcpy(errt[err_ptr].nm,token_buf) ;
                                    /* copy name of wrong identi.*/
  }
/*--------------------END ERROR----------------------------------*/

/*****************************************************************
 *                                                               *
 *                 ERRMESSAGE SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* prints the message of error that corresponds to the error found*/
```

```
errmessage(i)
int i ;                    /* i = error number */
 {                  /* err_ptr = global indicating error table index */
  printf("                                        ERROR   ") ;
  switch(i)
   {
    case 0 : printf(" 'INSERTION' expected\n") ;
             break ;
    case 1 : printf(" 'REPLACE' expected\n") ;
             break ;
    case 4 : printf(" '{' expected, ]s found\n",
                     errt[err_ptr].nm) ;
             break ;
    case 5 : printf(" '}' expected, ]s found\n",
                     errt[err_ptr].nm) ;
             break ;
    case 6 : printf(" 'INITIALIZE' expected,]s found\n",
                     errt[err_ptr].nm) ;
             break ;
    case 7 : printf(" 'PRINTOUT' expected, ]s found\n",
                     errt[err_ptr].nm) ;
             break ;
    case 25: printf(" name ]s is a keyword\n",
                     errt[err_ptr].nm) ;
             break ;
    case 26: printf(" count = ]d, >>EDITION discontinued\n",err_count);
             break ;
    case 27: printf(" '=' expected after ]s\n",
                     errt[err_ptr].nm) ;
             break ;
    case 28: printf(" ]s is undeclared\n",
                     errt[err_ptr].nm) ;
             break ;
    case 29: printf(" '(' expected after ]s\n",
                     errt[err_ptr].nm) ;
             break ;
    case 30: printf(" ]s is undefined function\n",
                     errt[err_ptr].nm) ;
             break ;
    case 31: printf(" ',' expected after ]s\n",
                     errt[err_ptr].nm) ;
              break ;
    case 32: printf(" ')' expected after ]s\n",
                     errt[err_ptr].nm) ;
              break ;
    case 33: printf(" unexpected EOF\n");
              break ;
    case 34: printf(" missing END\n") ;
             break ;
    case 35: printf(" incorrect # of args. on LHS ]s\n"
                     , errt[err_ptr].nm) ;
             break ;
    case 36: printf(" in syntax, ]s unrecognized \n",
                     errt[err_ptr].nm) ;
             break;
    case 37: printf(" count = 10, >>Edition discontinued\n");
             break ;
    case 38: printf(" = 0,  ***END OF EDITION***\n") ;
             break ;
    case 39: printf(" 1st DELAY index is > lim\n");
             break ;
    case 40: printf(" 2nd DELAY index is > lim\n");
             break ;
    case 41: printf(" missing {\n") ;
             break ;
    case 42: printf(" missing INITIALIZE\n") ;
             break ;
    case 43: printf(" missing ';'\n") ;
             break ;
    case 44: printf(" undefined function\n") ;
```

```
            break ;
    case 46: printf(" missing }\n") ;
            break ;
    case 47: printf(" missing ')'\n");
            break;
    case 49: printf(" incorrect # of input arguments in call\n");
            break ;
    case 50: printf(" incorrect # of out arguments in call\n");
            break ;
   }
  err_count = err_count + 1 ;
  if (err_count > 9)
   {
    error(37) ;
    outerror() ;
    exit(0) ;
   }
 }
/*--------------------END ERRMESSAGE----------------------------*/

/*****************************************************************
 *                                                               *
 *                      PRIM  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* verify what primitive will be used                            */

 prim(code1, code2)
int code1, code2;
 {
  int j ;
  /* savid[] saves symbol table indexes while savn[] saves desc. */
  /* numbers for output list names                               */
    getid(rp,33) ;          /*  function name                    */
    if (delimiter != 6)     /*  '(' should follow function name  */
      error(29) ;
    if (err_count == 0)
     {
      if (outpar > 0)       /* if # of outputs > 1, connect exten- */
       {                    /* sion pointers.                      */
        for (j = 0; j < outpar; j = j + 1)
         {
          fprintf(t4, "1 ]d 15 ]d ",savn[j], savn[j+1]) ;
          fprintf(t4,"1 ]d 16 ]d ",savn[j+1], savn[0]) ;
          symt[valact].despos = symt[valact].despos + 8 ;
          fadvance(8,t4) ;
         }
       }
     }
    findprim() ;
    if (primid >= prim_count)
     {
      findid() ;                /* function name is a type */
      primid = symt[symid].funcno ;
     }
    inptab[code1].inp_num = primt[primid].numpar ;
    if (err_count == 0)
       {
          fprintf(t4,"33 ]d ]d ",savn[0],primid) ;
          symt[valact].despos = symt[valact].despos + 3 ;
       }
    fadvance(3,t4) ;
    if (primt[primid].outp != (outpar+1))
      error(35) ;           /* # of outputs should be as in table  */
    for(j = 0; j <= outpar; j = j + 1)   /* update symbol table  */
     {                                   /* and desc table       */
      symt[(savid[j])].funcno = primid ;
      updesct(token_buf, savid[j], code2) ;
      code2 = code2 + 1;
     }
```

```
 }
/*--------------------END PRIM-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                     STRCPY SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/*This subroutine performs the copy from one string to another   */

strcpy(s,t)             /* copies s = t  */
char *s, *t ;
 {
  while(*s++ = *t++)
   ;
 }
/*--------------------END STRCPY---------------------------------*/

/*****************************************************************
 *                                                               *
 *                     GETID  SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* This subroutine makes the search in the VOHL file to verify   /*
/* the next identifier that will be used                         */

getid(rx, ernm) /* finds the next id and returns it in token_buf */
int ernm ;      /* error number in case of EOF */
FILE *rx ;
 {
  int i , c, flag ;
  flag = 0 ;
  delimiter = -1 ;
  i = 0 ;
  while((delimiter < 1 ) || (flag == 0))
   {                      /* flag = 1 when some non blank char is */
                          /* read into token buffer               */
    c = fgetc(rx) ;
    buff[bb] = c ;        /* buff is the 80 character buffer for  */
    bb = bb + 1 ;         /* printing the entire line after it is */
    if (bb > 78)          /* read.     bb = index for buff        */
     {
      if (prinpflag == 1)
       {
    /*  printf(" ]-s\n",buff) ;    */
       }
      bb = 0 ;
     }
    switch(c)
     {
      case ' ' : delimiter = -1 ;
                 break ;
      case ',' : delimiter = 1 ;
                 break ;
      case ';' : delimiter = 2 ;
                 break ;
      case ':' : delimiter = 3 ;
                 break ;
      case '=' : delimiter = 4 ;
                 break ;
      case ')' : delimiter = 5 ;
                 break ;
      case '(' : delimiter = 6 ;
                 break ;
      case '\n': if (flag == 1)       /* flag = 1 indicates that */
                   delimiter = 7 ;    /* some non blank character*/
                                      /* was read into token_buf */
                 buff[bb-1] = '\0' ;
```

```
                if (prinpflag == 1)
                   {
              /*    printf("]-s\n",buff) ;     print the line and   */
                   }
                                              /* output errors in the */
                outerror() ;                  /* line if any.         */

                bb = 0 ;                      /* initialize line buff */
                break ;
      case EOF : printf("]-s\n",buff) ;
                error(ernm) ;
                error(33) ;              /* EOF error                 */
                outerror() ;
                exit(0) ;                /* abort the program         */
                break ;
      default  : flag = 1 ;
                delimiter = 0 ;
     }
    if (delimiter == 0 )
     {
      if (i <= 6 )
       {
        token_buf[i] = c ;
        i = i + 1 ;
       }
     }
   }
  token_buf[i] = '\0' ;
 }
/*--------------------END GETID-----------------------------------*/

/*********************************************************************
 *                                                                   *
 *                 FIND_TOKEN SUBROUTINE                             *
 *                                                                   *
 *********************************************************************/
/* Token = maxkey if a nonkeyword name is encountered. It is        */
/* equal to the index of the keyword in the keyword table           */

find_token()
 {
  int i ;
  for (i = 0; i < maxkey; i = i + 1)       /* sequential search  */
    if (strcmp (token_buf,keyword[i]) == 0 )
      break ;
  toknn = i ;         /* token = maxkey,   if match is not found  */
 }
/*--------------------END FIND_TOKEN------------------------------*/

/*********************************************************************
 *                                                                   *
 *                   FIND_KEY SUBROUTINE                             *
 *                                                                   *
 *********************************************************************/
/* To verify what is the function that the user want to use    */

find_key()
 {
  int i ;
  for (i = 0; i < maxk; i = i + 1)       /* sequential search  */
    if (strcmp (token_buf,key[i]) == 0 )
      break ;
  toknn = i ;         /* token = maxk,   if match is not found  */
 }
/*--------------------END FIND_KEY--------------------------------*/
```

```
/*****************************************************************
 *                                                               *
 *                      FCOPY  SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* copy one file in another                                      */

fcopy(rr, ww)
FILE *rr, *ww ;
{
 int c ;
 while ((c = getc(rr)) != EOF)
   putc(c,ww) ;
}
/*--------------------END FCOPY-------------------------------*/

/*****************************************************************
 *                                                               *
 *                     CONNECT SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* Circular list generation for the circuit. Previous list is    */
/* broken and new circular loop is made.                         */

 connect(f,savn,fwdp)
 int f ;      /* f is 0 or 1 */
 int savn[], fwdp ;       /* savn has desc # of output names */
 {
  int i ;
  if (err_count == 0)
   {
    /*-----------------------update fanld-------------------------*/
    for (i = 0; i < normcount; i = i + 1)  /* find name in nort  */
     if (strcmp(nort[i].nom,savbuf) == 0)
       break;
    if (i < normcount)      /* if over ride is used for norm load */
      symt[symid].fanld = symt[symid].fanld + nort[i].nmld ;
    else                    /* use default value from prim. lib   */
      symt[symid].fanld=symt[symid].fanld + primt[savprim].normld;
    /*-------------------------------------------------------------*/

    fprintf(t4,"2 ]d ", symt[symid].descno) ;
    fprintf(t4,"3 ]d ", symt[symid].descno) ;
                        /* save current pointer from the input   */
    fprintf(t4,"1 ]d 8 ]d ",symt[symid].descno, savn[fwdp]) ;
    fprintf(t4,"1 ]d 11 ]d ",symt[symid].descno,f);
    fprintf(t4,"1 ]d 9 ]d ", savn[fwdp], f) ;
    fprintf(t4,"1 ]d 12 ]d ", savn[fwdp], f) ;
                          /* complete the C-list                 */
    fprintf(t4," \n");
    symt[valact].despos = symt[valact].despos + 20 ;
    filecount = 0 ;
   }
 }
/*--------------------END CONNECT-----------------------------*/

/*****************************************************************
 *                                                               *
 *                     PARSEID SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* This subroutine verifies of what type is the next identifier  */
/* that will be used.                                            */

parseid(i)
 int i ;                  /* i = token number to be compared to  */
  {
   getid (rp,33) ;              /* find the next identifier */
   find_token() ;               /* find token number        */
   if (toknn == maxkey)         /* check if name != function*/
```

```
    {
     findprim() ;
     if (primid < prim_count)
      error(25) ;                    /* keyword found            */
    }
   if (toknn != i)                   /* identifier of type 'i'   */
    error(i) ;                       /* expected.                */
  }
/*--------------------END PARSEID-------------------------------*/

/*****************************************************************
 *                                                               *
 *                    FADVANCE SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* This subroutine advances a counter while doing the SIMDATA    */
/*file to allow the change of line.                              */

 fadvance(numm,rx)
  int numm ;
  FILE *rx ;
  {
   filecount = filecount+  numm ;
   if (filecount > 22)
    {
     filecount = 0 ;
     fprintf(rx," \n") ;
    }
  }
/*--------------------END FADVANCE------------------------------*/

/*****************************************************************
 *                                                               *
 *                     STRCMP  SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* returns zero if string s is equal to string t                 */

strcmp(s,t)
char s[], t[] ;
 {
  int i ;
  i = 0 ;
  while(s[i] == t[i])
    if (s[i++] == '\0')
      return (0) ;
  return(s[i] - t[i]) ;
 }
/*--------------------END STRCMP--------------------------------*/

/*****************************************************************
 *                                                               *
 *                    IDSTRING SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* <IDSTRING> => id ; | <IDSTRING> id ,                          */

 IDSTRING(code)          /* code = 0 for inputs,-1 for outputs    */
 {                       /* -2 for internals.                     */
    while (delimiter != 2)          /* delimiter = ;       */
     {
      parseid(maxkey) ;             /* name                */
      update(code,sym_count) ;      /* update symbol table */
                                    /* code = 0 for input  */
                                    /* -1 for output       */
                                    /* prim # for TYPE     */
      if (code == 0 )
        code_input(desc_no) ;       /* input code gen.     */

      if (code <= 0 )
```

```
        desc_no = desc_no + 1 ;

      new = new + 1;
     }
   }
/*--------------------END IDSTRING-------------------------------*/

/*****************************************************************
 *                                                               *
 *                    UPDESCT SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* This routine updates the descriptor table.                    */
/* s = function name (type), num = symbol table index            */

 updesct(s, num, code)
 char s[8] ;
 int num, code ;
 {
  strcpy(desct[code].fun, s) ;
  desct[code].dnum = num;
  if (code == dptr)
      dptr = dptr + 1 ;
 }
/*--------------------END UPDESCT--------------------------------*/


/*****************************************************************
 *                                                               *
 *                     UPDATE SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* This routine updates the symbol table. sym_count = symbol     */
/* table index, desc_no = descriptor number,                     */
/* typ = function type, 0 = input, -1 = output, -2 = internal    */

update(typ,code)
int typ, code;
 {
   symt[code].descno = desc_no ;
   symt[code].funcno = typ ;
   strcpy(symt[code].name, token_buf) ;
  }
/*--------------------END UPDATE---------------------------------*/


/*****************************************************************
 *                                                               *
 *                 CODE_INPUT SUBROUTINE                         *
 *                                                               *
 *****************************************************************/
/* generates the code for the input descriptors                  */

code_input(i)
int i ;                    /* i = desc_no */
 {
 int j;
  if (err_count == 0)
   {
    fprintf(t4,"33 ]d 0 ",i) ;
    j=hashf(token_buf);
    fprintf(t4,"1 ]d 0 ]d ",i, j) ;
              /* parameters 0 and 1 contain the input line name */
    fprintf(t4,"1 ]d 2 1 ",i) ;
    fadvance(15,t4) ;
   }
 }
/*--------------------END CODE_INPUT-----------------------------*/
```

```
/*****************************************************************
 *                                                               *
 *                      HASHF  SUBROUTINE                        *
 *                                                               *
 *****************************************************************/

hashf(s)    /* finds hash value for string s  */
char *s;
{
 int hashval ;
 for (hashval = 0; *s != '\0' ;)
    hashval += *s++ ;                  /* convert from string to # */
 test(&hashval);                       /* and test for collision */
 hashtable[hashcount]=hashval;
 hashcount++;
 return (hashval) ;
}
/*--------------------END HASHF-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                       TEST  SUBROUTINE                        *
 *                                                               *
 *****************************************************************/

test(value)
int *value;
{
   int i;
   for (i=0; i<hashcount; i++)
      {
      if (hashtable[i]==(*value))          /* hashtable collision? */
         {
         (*value)=(*value)+11;             /* yes, add a prime number..*/
         test(value);                      /* and test again         */
         }
      }
}
/*--------------------END TEST------------------------------------*/


/*****************************************************************
 *                                                               *
 *                      CMODE  SUBROUTINE                        *
 *                                                               *
 *****************************************************************/
/* Code generator for mode. Code is generated only if mode != 0  */

 cmode(num, fan1, over1, tech1)
 int num, fan1, over1, tech1 ;
 {
  if (num <= (fan1 + over1))
    {
     if (tech1 == 16)
       fprintf(t5,"1 ]d 6 2 ",symt[descid].descno) ;
     else
       fprintf(t5,"1 ]d 6 1 ",symt[descid].descno) ;
    }
  else
    {
     if (tech1 == 16)
       fprintf(t5,"1 ]d 6 3 ",symt[descid].descno) ;
     else
       fprintf(t5,"1 ]d 6 4 ",symt[descid].descno) ;
     fadvance(4,t5) ;
    }
 }
/*--------------------END CMODE-----------------------------------*/
```

```
/*****************************************************************
 *                                                               *
 *                    MATGEN SUBROUTINE                          *
 *                                                               *
 *****************************************************************/
/* Also generates default mode values for all functions involved */

matgen(code,code1)
int code, code1;
{
 int par1, par2, i, j, k, l, m, n ;
 int num, f1, o1, t2;
 char s[8];


 /*----------------------DEFAULT MODE GENERATION----------------*/
 for (i = 0; i < sym_count; i = i + 1)
  {
   if (symt[i].descno >= 0)
    {
     if (symt[i].funcno > 0)
      {
       descid = i ;      /* cmode() needs descid for code gen.   */
       num = symt[i].fanld ;
       f1 = primt[(symt[i].funcno)].fanout ;
       o1 = primt[(symt[i].funcno)].overld ;
       t2 = primt[(symt[i].funcno)].technology ;
       if (num > f1)     /* if non zero mode */
         {
            cmode(num, f1, o1, t2) ;
            symt[i].delpos = symt[i].delpos + 4 ;
         }
      }
    }
  }
 /*-------------------------------------------------------------*/

 /*-------------------DEFAULT DELAYS AND MATRIX STRUCTURE-------*/
 i = code ;
 while( i < code1)
  {
   j = symt[desct[i].dnum].descno ;       /* descriptor number */
   k = symt[desct[i].dnum].funcno ;       /* function number   */
   n = desct[i].dnum ;
   symt[n].delpos = 0 ;
   strcpy(s, primt[k].nam2) ;    /* s contains name of function */
   bdread(s) ;      /* read block delays for s in rd/fdmat     */
   if (k >= 0)       /* if not input or types etc.*/
    {
     par1 = primt[k].numpar ;
     par2 = primt[k].outp   ;
     i =  i + par2 - 1 ;
         /* par1 = number of inputs, par2 = number of outputs  */
     for (l = 1; l <= par1; l = l + 2)   /* vertical scanning   */
      {
       if (l >  2)
        {
         if (l > 4)
           {
              fprintf(t5,"7 ");
              symt[n].delpos = symt[n].delpos + 1 ;
           }
         else
           {
              fprintf(t5,"6 ]d ",j) ;
              symt[n].delpos = symt[n].delpos + 2 ;
           }
        }
       else           /* inputs = 1 or 2 */
         {
```

```
        fprintf(t5,"4 ]d ",j) ;
        symt[n].delpos = symt[n].delpos + 2 ;
    }
fadvance(2,t5) ;
/*-----------------code for block delays-----------------*/
if (rdmat[l-1][0] != -1)
    {
        fprintf(t5,"5 2 ]d ",rdmat[l-1][0]) ;
        symt[n].delpos = symt[n].delpos + 3 ;
    }
if (fdmat[l-1][0] != -1)
    {
        fprintf(t5,"5 3 ]d ",fdmat[l-1][0]) ;
        symt[n].delpos = symt[n].delpos + 3 ;
    }
if ((l+1) <= par1)
  {
   if (rdmat[l][0] != -1)
      {
         fprintf(t5,"5 4 ]d ",rdmat[l][0]) ;
         symt[n].delpos = symt[n].delpos + 3 ;
      }
   if (fdmat[l][0] != -1)
      {
         fprintf(t5,"5 5 ]d ",fdmat[l][0]) ;
         symt[n].delpos = symt[n].delpos + 3 ;
      }
  }
fadvance(21,t5) ;
/*-------------------------------------------------------*/
for (m = 2; m <= par2 ; m = m + 1)
  {
   if (m == 2)
     {
      fprintf(t5,"14 ]d ",matcount) ;
      symt[n].delpos = symt[n].delpos + 2 ;
      matcount = matcount + 1 ;
     }
   else       /* if number of outputs > 2 */
     {
      fprintf(t5,"15 ]d ]d ", matcount - 1, matcount) ;
      symt[n].delpos = symt[n].delpos + 3 ;
      matcount = matcount + 1 ;
     }
   fadvance(3,t5) ;
   /*-----------------code for block delays---------------*/
   if (rdmat[l-1][m-1] != -1)
      {
         fprintf(t5,"16 ]d ]d ",matcount-1,rdmat[l-1][m-1]) ;
         symt[n].delpos = symt[n].delpos + 3 ;
      }
   if (fdmat[l-1][m-1] != -1)
      {
         fprintf(t5,"17 ]d ]d ",matcount-1,fdmat[l-1][m-1]) ;
         symt[n].delpos = symt[n].delpos + 3 ;
      }
   if ((l+1) <= par1)
     {
      if (rdmat[l][m-1] != -1)
        {
           fprintf(t5,"18 ]d ]d ",matcount-1,rdmat[l][m-1]) ;
           symt[n].delpos = symt[n].delpos + 3 ;
        }
      if (fdmat[l][m-1] != -1)
        {
           fprintf(t5,"19 ]d ]d ",matcount-1,fdmat[l][m-1]) ;
           symt[n].delpos = symt[n].delpos + 3 ;
        }
     }
   fadvance(12,t5) ;
```

```
        /*-------------------------------------------------------*/
      }    /* end for m = -- */
    }      /* end for l = -- */
  }        /* end if k >= 0  */
  i = i + 1 ;
}          /* end for i = -- */
}
/*--------------------END MATGEN-----------------------------------*/

/*****************************************************************
 *                                                               *
 *                    RFDEL  SUBROUTINE                          *
 *                                                               *
 *****************************************************************/

rfdel(n1,code1,code2,rx)
int n1, code1, code2;
FILE *rx ;
{
 int inp ;
 lim = 0 ;
 del_tab[index1].val = 0 ;
 index = index + 1 ;
 del_tab[index1].dsc_nb = target ;
 del_tab[index1].num_ipt = savpar1;
 del_tab[index1].num_opt = savpar2;
 lim = lim + parm2 - 1 ; /* lim is incremented by # of outputs*/
 fadvance(2) ;
 if (code1 > 1)          /* first access desc.   */
   {
     fprintf(rx,"6 ]d ",target);
     del_tab[index1].val = del_tab[index1].val + 2 ;
     del_tab[index1].indx = index;
     if (n1 == 0)
           del_tab[index1].typ_num = 10 ;
     else
           del_tab[index1].typ_num = 11 ;
     code1 = code1 - 2 ;
     while( code1 > 0)
      {
       fprintf(rx,"7 ");
       del_tab[index1].val = del_tab[index1].val + 1 ;
       fadvance(1,rx) ;
       code1 = code1 - 2 ;
      }
   }    /* if par1 > 1 */
  else                    /* if par1 = 1          */
   {
      fprintf(rx,"4 ]d ",target);
      del_tab[index1].val = del_tab[index1].val + 2 ;
      del_tab[index1].indx = index;
      if (n1 == 0)
            del_tab[index1].typ_num = 10 ;
      else
            del_tab[index1].typ_num = 11 ;
   }
  inp = savpar1 ] 2 ;   /* even or odd par1     */
  if (code2 == 0)
    {
       fprintf(rx,"5 ]d ]d ", ((2+n1) + 2*inp), num) ;
       del_tab[index1].val = del_tab[index1].val + 3 ;
    }
  else                    /* multi-output case    */
   {
    code2 = code2 - 1 ;
    fprintf(rx,"8 ") ;
    del_tab[index1].val = del_tab[index1].val + 1 ;
    del_tab[index1].indx = index;
    if (n1 == 0)
```

```
            del_tab[index1].typ_num = 10 ;
        else
            del_tab[index1].typ_num = 11 ;
        fadvance(3,rx) ;
        while (code2 > 0)
          {
           fprintf(rx,"9 ") ;
           del_tab[index1].val = del_tab[index1].val + 1 ;
           fadvance(1,rx) ;
           code2 = code2 - 1 ;
          }
        if (inp  == 0)
          {
           if (n1 == 0)
             {
                fprintf(rx,"10 ]d ",num) ;
                del_tab[index1].val = del_tab[index1].val + 2 ;
             }
           else
             {
                fprintf(rx,"11 ]d ",num) ;
                del_tab[index1].val = del_tab[index1].val + 2 ;
             }
          }
        else
          {
           if (n1 == 0)
             {
                fprintf(rx,"12 ]d ",num) ;
                del_tab[index1].val = del_tab[index1].val + 2 ;
             }
           else
             {
                fprintf(rx,"13 ]d ",num) ;
                del_tab[index1].val = del_tab[index1].val + 2 ;
             }
           fadvance(2,rx) ;
          }
       }
     code1 = savpar1 ;
     code2 = savpar2 ;
     index1 = index1 + 1 ;
  }      /* end RFDEL */
/*----------------------END RFDEL------------------------------*/
```

APPENDIX F
FILES NECESSARY TO THE CADD PROGRAMS

```
/*************************************************************************
 *                                                                       *
 *                        BLDEL file                                     *
 *                                                                       *
 *************************************************************************/
AND
2
0 0 1 1
1 0 1 1
OR
2
0 0 1 1
1 0 1 1
NAND
2
0 0 1 1
1 0 1 1
NOR
2
0 0 1 1
1 0 1 1
INVERT
1
0 0 1 1
ANDTHRE
3
0 0 1 1
1 0 1 1
2 0 1 1
NANDTHR
3
0 0 1 1
1 0 1 1
2 0 1 1
SRBLOCK
8
0 0 2 2
2 0 2 2
0 1 2 2
2 1 2 2
1 1 2 2
1 2 2 2
0 2 2 2
2 2 2 2
RETDBLO
24
0 0 2 2
2 0 2 2
3 0 2 2
4 0 2 2
5 0 2 2
0 1 2 2
2 1 2 2
3 1 2 2
4 1 2 2
5 1 2 2
0 2 2 2
1 2 2 2
2 2 2 2
3 2 2 2
4 2 2 2
```

```
0 3 2 2
2 3 2 2
3 3 2 2
4 3 2 2
0 4 2 2
2 4 2 2
3 4 2 2
4 4 2 2
5 4 2 2
ANDFOUR
4
0 0 1 1
1 0 1 1
2 0 1 1
3 0 1 1
NANDFOU
4
0 0 1 1
1 0 1 1
2 0 1 1
3 0 1 1
ORTHREE
3
0 0 1 1
1 0 1 1
2 0 1 1
ORFOUR
4
0 0 1 1
1 0 1 1
2 0 1 1
3 0 1 1
EXOR
2
0 0 1 1
1 0 1 1
END
/************************END BLDEL************************************/


/*************************************************************************
 *                                                                       *
 *                        BLOCK file                                     *
 *                                                                       *
 *************************************************************************/
/*-------------------------NAND GATE-----------------------------------*/
NAND(flg ,inpx,ou)
int flg , *inpx, *ou ;
{
 int i ;

 if (flg == 0)
   indata ( inpx, 2) ;

 i = (*inpx) + (*(inpx + 1)) ;
 switch(i)
  {
   case 0 : (*ou) = 1 ;
            break ;
   case 1 : (*ou) = 1 ;
            break ;
   case 2 : if ((*inpx) == 1)
              (*ou) = 0 ;
            else
              (*ou) = 1 ;
            break ;
    default : (*ou) = 2 ;
  }
 num_outs = 1 ;
}
```

```
/*-----------------------------------------------------------------*/

/*-------------------------NOR  GATE-------------------------------*/
NOR(flg ,inpx, ou)
int flg , *inpx, *ou ;
{
 int i ;

 if (flg == 0)
   indata ( inpx, 2) ;

 i = (*inpx) + (*(inpx + 1)) ;
 switch(i)
  {
   case 0 : (*ou) = 1 ;
            break ;
   case 1 : (*ou) = 0 ;
            break ;
   case 2 : if ((*inpx) == 1)
              (*ou) = 0 ;
            else
              (*ou) = 2 ;
            break ;
    default : if (((*inpx) == 1) || ((*(inpx + 1)) == 1))
                (*ou) = 0;
              else
                (*ou) = 2 ;
  }
 num_outs = 1 ;
}
/*-----------------------------------------------------------------*/

/*---------------------THREE INPUT AND GATE----------------------*/
ANDTHRE(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int inn[2] ;
  if (flg == 0)
   indata(inpx, 3) ;

  AND(1,inpx, inn) ;
  inn[1] = (*(inpx + 2)) ;
  AND(1,inn, ou) ;

  num_outs = 1 ;
 }
/*-----------------------------------------------------------------*/

/*---------------------THREE INPUT NAND GATE---------------------*/
NANDTHR(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int m ;

  if (flg == 0)
    indata(inpx, 3) ;

  ANDTHRE(1,inpx, &m) ;
  INVERT(1,&m, ou) ;

  num_outs = 1 ;
 }
/*-----------------------------------------------------------------*/

/*---------------------SRBLOCK-----------------------------------*/
SRBLOCK(flg ,inpx, ou)
```

```
int flg , *inpx, *ou ;
 {
  int inn[2] ;

  if (flg == 0)
    indata(inpx, 3) ;

  NAND(1, inpx, inn) ;
  inn[1] = (*(inpx + 2)) ;
  (*ou) = (*inn) ;
  NAND(1, inn, (ou+1)) ;
  (*(ou+2)) = (*(ou + 1)) ;      /* X output is same as Q'  */

  num_outs = 3 ;
 }
/*-------------------------------------------------------------------*/


/*-------------------------RETDBLO()---------------------------*/
RETDBLO(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int i, inn[3] ;

  if (flg  == 0)
    indata(inpx, 6) ;

  NAND(1,(inpx + 3),inn) ;        /* A =  NAND(X1, X2) */
                                  /* inn[0] = A        */
  inn[1] = (*inpx) ;              /* inn[1] = clr      */
  inn[2] = (*(inpx + 2)) ;        /* inn[2] = CLK      */
  NANDTHR(1,inn, (ou+3));         /* X2 = NANDTHRE(A,clr,CLK);  */
  NANDTHR(1,(inpx + 2),&(inn[2]));/* B = NANDTHRE(X2, CLK, X1)*/
                                  /* inn[2] = B                 */
  NAND(1,(inpx + 4),ou) ;         /* Q = NAND(X2, X3) ;         */
  inn[0] = (*ou) ;                /* inn[0] = Q                 */
  inn[1] = (*inpx) ;              /* inn[1] = clr               */
  NANDTHR(1, inn,(ou+1));         /* Qc = NANDTHRE(Q,clr,B);    */
  inn[0] = (*(inpx + 1)) ;        /* inn[0] = D                 */
  NANDTHR(1, inn,(ou+2));         /* X1 = NANDTHRE(B, clr, D)   */
  (*(ou+4)) = (*(ou + 1)) ;       /* X3 = Qc                    */
  num_outs = 5 ;
 }
/*-------------------------------------------------------------------*/


/*--------------------FOUR  INPUT AND GATE----------------------*/
ANDFOUR(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int inn[2];

  if (flg == 0)
    indata(inpx, 4) ;

  ANDTHRE(1, inpx, inn) ;
  inn[1] = (*(inpx + 3)) ;
  AND(1, inn, ou) ;

  num_outs = 1 ;
 }
/*-------------------------------------------------------------------*/


/*----------------------NANDFOUR()-----------------------------*/
NANDFOU(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int m ;
```

```
  if (flg == 0)
    indata(inpx, 4) ;

  ANDFOUR(1,inpx, &m) ;
  INVERT(1, &m, ou) ;

  num_outs = 1 ;
 }
/*-------------------------------------------------------------------*/


/*-------------------------ORTHREE()---------------------------*/
ORTHREE(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int inn[2];

  if (flg == 0)
    indata(inpx, 3) ;

  OR(1,inpx, inn) ;
  inn[1] = (*(inpx + 2)) ;
  OR(1, inn, ou) ;

  num_outs = 1 ;
 }
/*-------------------------------------------------------------------*/


/*--------------------FOUR INPUT OR GATE-----------------------*/
ORFOUR(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int inn[2] ;

  if (flg == 0)
    indata(inpx, 4) ;

  ORTHREE(1,inpx, inn) ;
  inn[1] = (*(inpx + 3)) ;
  OR(1, inn, ou) ;

  num_outs = 1 ;
 }
/*-------------------------------------------------------------------*/


/*-------------------------EXOR GATE---------------------------------*/
EXOR(flg ,inpx, ou)
int flg , *inpx, *ou ;
{
 int i ;

 if (flg == 0)
   indata(inpx, 2) ;

 i = (*inpx) + (*(inpx + 1)) ;
 switch(i)
  {
   case 0 : (*ou) = 0 ;
            break ;
   case 1 : (*ou) = 1 ;
            break ;
   case 2 : if ((*inpx) == 1)
              (*ou) = 0 ;
            else
              (*ou) = 2 ;
            break ;
    default : (*ou) = 2 ;
```

```
  }
 num_outs = 1 ;
}
/*-----------------------------------------------------------------*/
/************************END BLOCK*************************************/

/*************************************************************************
 *                                                                       *
 *                        FADDR file                                     *
 *                                                                       *
 *************************************************************************/
/* Addresses of primitives supported */
 pnfn[3] = NAND ;
 pnfn[4] = NOR ;
 pnfn[6] = EXOR ;
 pnfn[7] = ANDTHRE ;
 pnfn[8] = NANDTHR ;
 pnfn[9] = SRBLOCK ;
 pnfn[10]= RETDBLO ;
 pnfn[11]= ANDFOUR ;
 pnfn[12]= NANDFOU ;
 pnfn[13]= ORTHREE ;
 pnfn[14]= ORFOUR  ;
 pncnt = 15 ;
/************************END FADDR*************************************/

/*************************************************************************
 *                                                                       *
 *                        FTYPE file                                     *
 *                                                                       *
 *************************************************************************/
int NAND() ;
int NOR() ;
int ANDTHRE() ;
int ORTHREE() ;
int NANDTHR() ;
int SRBLOCK() ;
int RETDBLO() ;
int ANDFOUR() ;
int NANDFOU() ;
int ORFOUR() ;
int EXOR() ;
/************************END FTYPE*************************************/

/*************************************************************************
 *                                                                       *
 *                      PRIM2.C file                                     *
 *                                                                       *
 *************************************************************************/

/*************************************************************************
      Logic Primitive Descriptions File
*************************************************************************/
#include "c:\lc\stdio.h"

#define maxprim 100

struct prim_tab {
     char nam[8],nam2[8];
     int numpar,outp;
     int normld, fanout;
     int technology, overld;
     };
struct prim_tab *tempptr;

primsetup(primptr)
   struct prim_tab *primptr;
```

```
 {
 extern int prim_count,features[maxprim][2];
 int i;
 char ch;
 FILE *xp;

tempptr=primptr;
strcpy((*primptr).nam, "READIN") ;
primptr++;
strcpy((*primptr).nam, "AND") ;
primptr++;
strcpy((*primptr).nam, "OR") ;
primptr++;
strcpy((*primptr).nam, "NAND") ;
primptr++;
strcpy((*primptr).nam, "NOR") ;
primptr++;
strcpy((*primptr).nam, "INVERT") ;
primptr++;
strcpy((*primptr).nam, "EXOR") ;
primptr++;
strcpy((*primptr).nam, "ANDTHRE") ;
primptr++;
strcpy((*primptr).nam, "NANDTHR") ;
primptr++;
strcpy((*primptr).nam, "SRBLOCK") ;
primptr++;
strcpy((*primptr).nam, "RETDBLO") ;
primptr++;
strcpy((*primptr).nam, "ANDFOUR") ;
primptr++;
strcpy((*primptr).nam, "NANDFOU") ;
primptr++;
strcpy((*primptr).nam, "ORTHREE") ;
primptr++;
strcpy((*primptr).nam, "ORFOUR") ;
primptr++;

strcpy((*primptr).nam,"USER1");       /* set up user-defined prims */
primptr++;
strcpy((*primptr).nam,"USER2");       /* we'll check to see if any */
primptr++;
strcpy((*primptr).nam,"USER3");      /* are actually present later */
primptr++;
strcpy((*primptr).nam,"USER4");
primptr++;
strcpy((*primptr).nam,"USER5");
primptr++;
strcpy((*primptr).nam,"USER6");
primptr++;
strcpy((*primptr).nam,"USER7");
primptr++;
strcpy((*primptr).nam,"USER8");
primptr++;
strcpy((*primptr).nam,"USER9");
primptr++;
strcpy((*primptr).nam,"USER10");
primptr++;
strcpy((*primptr).nam,"USER11");
primptr++;
strcpy((*primptr).nam,"USER10");
primptr++;
strcpy((*primptr).nam,"USER12");
primptr++;
strcpy((*primptr).nam,"USER13");
primptr++;
strcpy((*primptr).nam,"USER14");
primptr++;
strcpy((*primptr).nam,"USER15");
primptr++;
```

```
  strcpy((*primptr).nam,"USER16");
  primptr++;
  strcpy((*primptr).nam,"USER17");
  primptr++;
  strcpy((*primptr).nam,"USER18");
  primptr++;
  strcpy((*primptr).nam,"USER19");
  primptr++;
  strcpy((*primptr).nam,"USER20");
  primptr++;
  strcpy((*primptr).nam,"USER21");
  primptr++;
  strcpy((*primptr).nam,"USER22");
  primptr++;
  strcpy((*primptr).nam,"USER23");
  primptr++;
  strcpy((*primptr).nam,"USER24");
  primptr++;
  strcpy((*primptr).nam,"USER25");
  primptr++;
  strcpy((*primptr).nam,"USER26");
  primptr++;
  strcpy((*primptr).nam,"USER27");
  primptr++;
  strcpy((*primptr).nam,"USER28");
  primptr++;
  strcpy((*primptr).nam,"USER29");
  primptr++;
  strcpy((*primptr).nam,"USER30");
  primptr++;
  strcpy((*primptr).nam,"USER31");
  primptr++;
  strcpy((*primptr).nam,"USER30");
  primptr++;
  strcpy((*primptr).nam,"USER32");
  primptr++;
  strcpy((*primptr).nam,"USER33");
  primptr++;
  strcpy((*primptr).nam,"USER34");
  primptr++;
  strcpy((*primptr).nam,"USER35");
  primptr++;
  strcpy((*primptr).nam,"USER36");
  primptr++;
  strcpy((*primptr).nam,"USER37");
  primptr++;
  strcpy((*primptr).nam,"USER38");
  primptr++;
  strcpy((*primptr).nam,"USER39");
  primptr++;
  strcpy((*primptr).nam,"USER40");

  primptr=tempptr;                    /* reset pointer to start */

  xp=fopen("D:primitiv.dat","r");     /*  get additional info from disk*/
  fscanf(xp,"]d",&prim_count);
  for (i=0; i<prim_count; i++)
     {
     fscanf(xp,"]s ]d ]d ]d ]d",(*primptr).nam2,&(*primptr).numpar,
                    &(*primptr).outp,&features[i][0],&features[i][1]);
     primptr++;
     }
  fclose(xp);

  }
/************************END PRIM2.C*********************************/
```

```
/*************************************************************************
 *                                                                       *
 *                    PRIMITIV.DAT file                                  *
 *                                                                       *
 *************************************************************************/
15
 READIN 0 0 2 0
 AND 2 1 2 0
 OR 2 1 2 0
 NAND 2 1 2 0
 NOR 2 1 2 0
 INVERT 1 1 2 0
 EXOR 2 1 2 0
 ANDTHRE 3 1 2 0
 NANDTHR 3 1 2 0
 SRBLOCK 3 3 2 1
 RETDBLO 6 5 2 1
 ANDFOUR 4 1 2 0
 NANDFOU 4 1 2 0
 ORTHREE 3 1 2 0
 ORFOUR 4 1 2 0
/************************END PRIMITIV.DAT*****************************/


/*************************************************************************
 *                                                                       *
 *                          STRUC  file                                  *
 *                                                                       *
 *************************************************************************/
MODULE : INVERT ;
INPUTS : A ;
OUTPUTS : B ;
TYPES: ;
{
0;
}
MODULE : AND ;
INPUTS : A, B ;
OUTPUTS : Z ;
TYPES: ;
{
0;
}
MODULE : OR ;
INPUTS : A, B ;
OUTPUTS : Z ;
TYPES : ;
{
0;
}
MODULE : ANDTHREE ;
INPUTS : A, B, C ;
OUTPUTS : Z ;
TYPES : ;
{
0;
}
MODULE: NANDTHRE ;
INPUTS: A, B, C ;
OUTPUTS : Z ;
TYPES : ;
{
0;
}
MODULE : ORTHREE ;
INPUTS : A, B, C ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
```

```
}
MODULE : EXOR ;
INPUTS : A, B ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : NAND  ;
INPUTS : A, B ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : NOR ;
INPUTS : A, B ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : ANDFOUR ;
INPUTS : A, B, C, D ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : NANDFOUR ;
INPUTS : A, B, C, D ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : ORFOUR ;
INPUTS : A, B, C, D ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : SRBLOCK ;
INPUTS : S, R, X ;
OUTPUTS : Q, QC, Y ;
TYPES : ;
{
 Q = NAND(S, QC) ;
 QC = NAND(Q, R) ;
}
MODULE : RETDBLO ;
INPUTS : CLR, D, CLK, DUM1, DUM2, DUM3 ;
OUTPUTS : Q, QC, DM1, DM2, DM3 ;
TYPES : INTERNALS : X, Y, W, Z ;
{
 X = NAND(Z, Y) ;
 Y = NANDTHRE( X,CLR, CLK) ;
 W = NANDTHRE(Y, CLK, Z) ;
 Z = NANDTHRE(W, CLR, D) ;
 Q = NAND(Y, QC) ;
 QC = NANDTHRE(Q, CLR, W) ;
}
/************************END STRUC************************************/
```

```
/*****************************************************************
 *                                                               *
 *                      COMP2B.BAT file                          *
 *                                                               *
 *****************************************************************/
lc -ml -n20 cadd2b
lc -ml -n20 prim2
lc -ml -n20 precomp
link \lc\l\cl+cadd2b+prim2+precomp,caddrd2b,,\lc\l\lcl
/************************END COMP2B.BAT*****************************/


/*****************************************************************
 *                                                               *
 *                      COMP3B.BAT file                          *
 *                                                               *
 *****************************************************************/
lc -ml -n20 cadd3b
lc -ml -n20 prim2
lc -ml -n20 prcmp1
link \lc\l\cl+cadd3b+prim2+prcmp1,caddrd3b,,\lc\l\lcl
/************************END COMP3B.BAT*****************************/


/*****************************************************************
 *                                                               *
 *                    MODEL2A.BAT file                           *
 *                                                               *
 *****************************************************************/
echo off
echo                          Setting up...
if ]2==c goto compl
if ]2==C goto compl
if ]2==S goto simul
if ]2==s goto simul
copy ]1 d:\
copy ]1.in d:\
copy \simd\struc d:
copy \simd\bldel d:
copy \simd\primitiv.dat d:
echo                  Entering the VOHL Compiler...
caddrd2b =16000 d:]1
simrd D:]1.in D:]1.out
type d:]1.out
copy d:]1.out c:\simd\]1.out
pause
goto done
:compl
copy ]1 d:\
copy \simd\struc d:
copy \simd\bldel d:
copy \simd\primitiv.dat d:
echo                  Entering the VOHL Compiler...
caddrd2b =16000 d:]1
copy d:modf c:\simd\]1.mdf
copy d:initi c:\simd\]1.int
copy d:descf c:\simd\]1.dcf
copy d:delf c:\simd\]1.ddf
copy d:prnt c:\simd\]1.ptt
goto done
:simul
copy ]1.in d:\
copy c:\simd\]1.mdf d:modf
copy c:\simd\]1.int d:initi
copy c:\simd\]1.dcf d:descf
copy c:\simd\]1.ddf d:delf
copy c:\simd\]1.ptt d:prnt
simrd1 D:]1.in D:]1.out
type d:]1.out
copy d:]1.out c:\simd\]1.out
```

```
pause
:done
echo                          Cleaning Up....
copy d:modf c:\simd\]1.mdf
copy d:initi c:\simd\]1.int
copy d:descf c:\simd\]1.dcf
copy d:delf c:\simd\]1.ddf
copy d:prnt c:\simd\]1.ptt
copy d:symtable c:\simd\]1.stb
copy d:descptab c:\simd\]1.dct
copy d:nortable c:\simd\]1.nrt
copy d:inptable c:\simd\]1.ipt
copy d:deltable c:\simd\]1.del
echo                    Saving Descriptor Data...
copy d:primitiv.dat \simd
copy d:struc \simd
echo                          Returning to DOS...
/************************END MODEL2A.BAT*****************************/


/*************************************************************************
 *                                                                       *
 *                     MODEL3B.BAT file                                  *
 *                                                                       *
 *************************************************************************/
echo off
echo                             Setting up...
copy \simd\]1.mdf d:modf
copy \simd\]1.int d:initi
copy \simd\]1.dcf d:descf
copy \simd\]1.ddf d:delf
copy \simd\]1.ptt d:prnt
copy \simd\]1.stb d:symtable
copy \simd\]1.dct d:descptab
copy \simd\]1.ipt d:inptable
copy \simd\]1.nrt d:nortable
copy \simd\]1.del d:deltable
if ]2==c goto compl
if ]2==C goto compl
if ]2==S goto simul
if ]2==s goto simul
copy ]1.ed d:\
copy ]1.in d:\
copy \simd\struc d:
copy \simd\bldel d:
copy \simd\primitiv.dat d:
echo                    Entering the VOHL Editor...
caddrd3a =16000 d:]1.ed
simrd1 D:]1.in D:]1.out
type d:]1.out
copy d:]1.out c:\simd\]1.out
pause
goto done
:compl
copy ]1.ed d:\
copy \simd\struc d:
copy \simd\bldel d:
copy \simd\primitiv.dat d:
echo                    Entering the VOHL Editor...
caddrd3a =16000 d:]1.ed
copy d:simdata c:\simd\]1.sim
goto done
:simul
copy ]1.in d:\
simrd1 D:]1.in D:]1.out
type d:]1.out
copy d:]1.out c:\simd\]1.out
pause
:done
echo                          Cleaning Up....
```

```
copy d:modf c:\simd\]1.mdf
copy d:initi c:\simd\]1.int
copy d:descf c:\simd\]1.dcf
copy d:delf c:\simd\]1.ddf
copy d:prnt c:\simd\]1.ptt
copy d:symtable c:\simd\]1.stb
copy d:descptab c:\simd\]1.dct
copy d:inptable c:\simd\]1.ipt
copy d:nortable c:\simd\]1.nrt
copy d:deltable c:\simd\]1.del
echo                  Saving Descriptor Data...
copy d:primitiv.dat \simd
copy d:struc \simd
echo                        Returning to DOS...
/************************END MODEL3B.BAT*****************************/
```

APPENDIX G
NECESSARY FILES TO THE MULTISIM PACKAGE

CADD2A.C - The source code for the VOHL compiler, that includes the code generator.

PRECOMP.C - The source code for the precompilation routines of the compiler program.

PRIM2.C - The primitive initialization routine.

CADD3B.C - The source code for the editor program.

PRCMP1.C - The source code for the second part of the editor program. This program needs to be linked with the CADD3B.C program to allow the edition of a circuit.

TWHEEL1.C - The source code for the timing wheel simulator program.

FTYPE - The primitive function type declarations. It is an #include file.

FADDR - The primitive function pointer initializations. It is an #include file.

BLOCK - The C-language behavioral descriptions for the primitives. It is an #include file.

STRUC - The primitive library containing VOHL structural descriptions.

PRIMITIV.DAT - The data file containing the user's names fo the primitives and various other information.

BLDEL - The block delays for each primitive. This file contains the value of the default delays from each input to each output for all the primitives of the system.

COMP2A.BAT - The DOS file that allows the compilation and link of the various programs that perform the compilation of a circuit.

COMP3B.BAT - The DOS file that allows the compilation and link of the various programs that perform the edition of a circuit.

MODEL2A.BAT - The DOS file that allows the execution of the compilation and simulation of a circuit.

MODEL3B.BAT - The DOS file that allows the execution of the edition and simulation of a circuit.

APPENDIX H
THE ALU CIRCUIT

```
MODULE : ALU ;
INPUTS : A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q,
         R, S, Y, Z ;
OUTPUTS : ALU3, ALU2, ALU1, ALU0, CYOUT, ZR, OV, RI, PI, GI ;
TYPES : INTERNALS : H0, H1, H2, H3, K0, K1, K2, K3, S0, W0, W1, W2, W3,
        S1, S2, S3, L0, L1, L2, L3, H4, H5, H6, H7, H8, H9, H10 ;
   {
    K3, K2, K1, K0, H5, H7, PI, GI, H0 = ARITHMETIC (I, A, B, C, D,
                                                     H, G, F, E, J, K) ;
    L3, L2, L1, L0, H1 = LOGIC (A, B, C, D, E, F, G, H, L, M, N, O, P) ;
    H4, S3, S2, S1, S0, H6, H2 = SHIFT (Y, A, B, C, D, Z, Q, R, S) ;
    W0 = ORTHREE (K0, L0, S0) ;
    W1 = ORTHREE (K1, L1, S1) ;
    W2 = ORTHREE (K2, L2, S2) ;
    W3 = ORTHREE (K3, L3, S3) ;
    H9 = ORFOUR (W0, W1, W2, W3) ;
    H10 = INVERT (H9) ;
    H8 = OR (H6, H5) ;
    H3 = ORTHREE (H0, H1, H2) ;
    ALU3, ALU2, ALU1, ALU0, CYOUT, ZR, OV, RI = 8GATE (W3, W2, W1, W0,
                                                        H8, H10, H7, H4, H3) ;
    }
DEFINE: ;
INITIALIZE: Z = 2 ;
PRINTOUT: CYOUT, ALU3, ALU2, ALU1, ALU0, RI, ZR, OV ;

MODULE : ARITHMETIC ;
INPUTS : X, A, B, C, D, M, N, O, L, W, Z ;
OUTPUTS : K3, K2, K1, K0, COUT, OV, PI, GI, ENA ;
TYPES : INTERNALS : G0, G1, G2, G3, G4, G5, G6, G7, G8, G9 ;
  {
    G0, G1, G2, G3 = ADDSUB4 (M, N, O, L, W) ;
    G4, G5, G6, G7, G8, PI, GI = 4BITADD(X, A, B, C, D, G0, G1, G2, G3);
    G9 = ADDOV (D, G3, G7) ;
    ENA = OR (W, Z) ;
    K3, K2, K1, K0, COUT, OV = 6GATE (G7, G6, G5, G4, G8, G9, ENA) ;
  }

MODULE : ADDSUB4 ;
INPUTS : M, N, O, P, X ;
OUTPUTS : M0, N0, O0, P0 ;
TYPES : ;
     {
       M0, N0 = ADDSUB2(M, N, X) ;
       O0, P0 = ADDSUB2(O, P, X) ;
     }

MODULE : ADDSUB2 ;
INPUTS : M, N, X ;
OUTPUTS : M0, N0 ;
TYPES : ;
     {
       M0 = EXOR(M, X) ;
       N0 = EXOR(N, X) ;
     }

MODULE : 4BITADD ;
INPUTS : X, A, B, C, D, M, N, O, Q ;
OUTPUTS : S0, S1, S2, S3, COUT, P, G ;
TYPES : INTERNALS : A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11,
                    A12, A13, A14, A15, A16, A17, A18, A19, A20 ;
```

```
      {
        S0, A17, A13 = ADD(X, A, M) ;
        A0 = AND(X, A17) ;
        A10 = OR(A0, A13) ;
        S1, A18, A14 = ADD(A10, B, N) ;
        A1 = AND(A0, A18) ;
        A2 = AND(A18, A13) ;
        A11 = ORTHREE(A1, A2, A14) ;
        S2, A19, A15 = ADD(A11, C, O) ;
        A3 = AND(A1, A19) ;
        A4 = AND(A2, A19) ;
        A5 = AND(A18, A14) ;
        A12 = ORFOUR(A3, A4, A5, A15) ;
        S3, A20, A16 = ADD(A12, D, Q) ;
        A6 = AND(A3, A20) ;
        A7 = AND(A4, A20) ;
        A8 = AND(A5, A20) ;
        A9 = AND(A20, A15) ;
        G = ORFOUR(A7, A8, A9, A16) ;
        COUT = OR(A6, G) ;
        P = ANDFOUR(A20, A19, A18, A17) ;
      }

MODULE : ADD ;
INPUTS : C, A, B ;
OUTPUTS : S, PI, GI ;
TYPES : ;
    {
     PI = EXOR(A, B) ;
     GI = AND(A, B) ;
     S = EXOR(C, PI) ;
    }

MODULE : ADDOV ;
INPUTS : A, B, X ;
OUTPUTS : OV ;
TYPES : INTERNALS : B0, B1, B2, B3, B4 ;
    {
     B0 = INVERT (X) ;
     B1 = INVERT (A) ;
     B2 = INVERT (B) ;
     B3 = ANDTHREE (A, B, B0) ;
     B4 = ANDTHREE (B2, B1, X) ;
     OV = OR (B3, B4) ;
    }

MODULE : LOGIC ;
INPUTS : A, B, C, D, M, N, O, P, K, Y, X, Z, R ;
OUTPUTS : L3, L2, L1, L0, ENL ;
TYPES : INTERNALS : M0, M1, M2, M3, M4 ;
     {
      M0, M1, M2, M3 = LGUNIT4 (A, B, C, D, M, N, O, P, K, Y, X, Z, R) ;
      M4 = ORFOUR (K, Y, X, Z) ;
      ENL = OR (M4, R) ;
      L0, L1, L2, L3 = 4GATE (M0, M1, M2, M3, ENL) ;
     }

MODULE : LGUNIT4 ;
INPUTS : A, B, C, D, M, N, O, P, K, Y, X, Z, R ;
OUTPUTS : L0, L1, L2, L3 ;
TYPES : ;
     {
      L0, L1 = LGUNIT2 (A, B, M, N, K, Y, X, Z, R) ;
      L2, L3 = LGUNIT2 (C, D, O, P, K, Y, X, Z, R) ;
     }

MODULE : LGUNIT2 ;
INPUTS : A, B, C, D, K, Y, X, Z, R ;
OUTPUTS : L0, L1 ;
TYPES : ;
```

```
    {
     L0 = LGUNIT1 (A, C, K, Y, X, Z, R) ;
     L1 = LGUNIT1 (B, D, K, Y, X, Z, R) ;
    }

MODULE : LGUNIT1 ;
INPUTS : A, B, K, Y, X, Z, R ;
OUTPUTS : L0 ;
TYPES : INTERNALS : C0, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10 ;
    {
     C0 = AND (A, B) ;
     C1 = OR (A, B) ;
     C2 = EXOR (A, B) ;
     C3 = INVERT (A) ;
     C4 = INVERT (B) ;
     C5 = AND (C0, K) ;
     C6 = AND (C1, Y) ;
     C7 = AND (C2, X) ;
     C8 = AND (C3, Z) ;
     C9 = AND (C4, R) ;
     C10 = ORFOUR (C5, C6, C7, C8) ;
     L0 = OR (C9, C10) ;
    }

MODULE : SHIFT ;
INPUTS : X, A, B, C, D, Y, M, N, O ;
OUTPUTS : RI, S3, S2, S1, S0, CY, ENS ;
TYPES : INTERNALS : F0, F1, F2, F3, F4, F5;
   {
     ENS = ORTHREE (M, N, O) ;
     F1, F2, F3, F4, F0, F5 = SHUNIT4 (Y, D, C, B, A, X, N, M, O) ;
     RI, S0, S1, S2, S3, CY = 6GATE (F0, F1, F2, F3, F4, F5, ENS) ;
   }

MODULE : SHUNIT4 ;
INPUTS : L, D, C, B, A, R, N, X, Y ;
OUTPUTS : S0, S1, S2, S3, OUTR, OUTL ;
TYPES : INTERNALS : D0, D1 ;
    {
     S0, S1, OUTR, D0 = SHUNIT2 (C, B, A, R, N, X, Y) ;
     S2, S3, D1, OUTL = SHUNIT2 (L, D, C, B, N, X, Y) ;
    }

MODULE : SHUNIT2 ;
INPUTS : L, B, A, R, N, X, Y ;
OUTPUTS : S0, S1, OUTR, OUTL ;
TYPES : ;
    {
     S0 = SHUNIT1 (B, A, R, N, X, Y) ;
     S1 = SHUNIT1 (L, B, A, N, X, Y) ;
     OUTR = AND (A, Y) ;
     OUTL = AND (B, X) ;
    }

MODULE : SHUNIT1 ;
INPUTS : C, B, A, N, L, R ;
OUTPUTS : S ;
TYPES : INTERNALS : E0, E1, E2 ;
     {
      E0 = AND (B, N) ;
      E1 = AND (A, L) ;
      E2 = AND (C, R) ;
      S = ORTHREE (E0, E1, E2) ;
     }

MODULE : 8GATE ;
INPUTS : A, B, C, D, E, F, G, H, Z ;
OUTPUTS : B0, B1, B2, B3, B4, B5, B6, B7 ;
TYPES : ;
   {
```

```
    B0, B1, B2, B3 = 4GATE (A, B, C, D, Z) ;
    B4, B5, B6, B7 = 4GATE (E, F, G, H, Z) ;
   }

MODULE : 6GATE ;
INPUTS : A, B, C, D, E, F, X ;
OUTPUTS : G0, G1, G2, G3, G4, G5 ;
TYPES : ;
    {
     G0, G1, G2, G3 = 4GATE (A, B, C, D, X) ;
     G4, G5 = 2GATE ( E, F, X) ;
    }

MODULE : 4GATE ;
INPUTS : A, B, C, D, X ;
OUTPUTS : I0, I1, I2, I3 ;
TYPES : ;
     {
      I0, I1 = 2GATE (A, B, X) ;
      I2, I3 = 2GATE (C, D, X) ;
     }

MODULE : 2GATE ;
INPUTS : A, B, X ;
OUTPUTS : N0, N1 ;
TYPES : ;
    {
      N0 = AND (A, X) ;
      N1 = AND (B, X) ;
    }

END;
```

LIST OF REFERENCES

1. Mahmood, Ausif *Development of a Multilevel Logic Simulator for VLSI Systems*, PhD. dissertation, Washington State University, August 1985, Chapter 4 "Multilevel Hardware Description Language Design and Implementation", pp. 62-88.

2. Kelly, J. Scott *MultiSimPC: A Multilevel Logic Simulator for Microcomputers*, MS thesis, Naval Postgraduate School, Monterey, California, September 1986, Chapter III "Microcomputer Implementation", Section C, pp. 43-45.

3. Mahmood, Ausif *Development of a Multilevel Logic Simulator for VLSI Systems*, PhD. dissertation, Washington State University, August 1985, Chapter 3 "Multilevel Simulator Design", Section D, pp. 40-50.

4. Kelly, J. Scott *MultiSimPC: A Multilevel Logic Simulator for Microcomputers*, MS thesis, Naval Postgraduate School, Monterey, California, September 1986, Chapter II "The MULTISIM Package", Section B, Sub-section 2, pp. 27-32.

5. Kelly, J. Scott *MultiSimPC: A Multilevel Logic Simulator for Microcomputers*, MS thesis, Naval Postgraduate School, September 1986, September 1986, Chapter II "The MULTISIM Package", Section B, Sub-section 3, pp. 33-34.

6. Mahmood, Ausif *Development of a Multilevel Logic Simulator for VLSI Systems*, PhD. dissertation, Washington State University, August 1985, Chapter 2 "Review of Basic Simulation Algorithms", Section C, pp. 16-18.

7. Mahmood, Ausif *Development of a Multilevel Logic Simulator for VLSI Systems*, PhD. dissertation, Washington State University, August 1985, Chapter 3 "Multilevel Simulator Design", Sections C and D, pp. 40-52.

8. Kelly, J. Scott *MultiSimPC: A Multilevel Logical Simulator for Microcomputers*, MS thesis, Naval Postgraduate School, Monterey, California, September 1986, Chapter III "Microcomputer Implementation", Sections B, C and D, pp. 43-55.

9. Kelly, J. Scott *MultiSimPC: A Multilevel Logical Simulator for Microcomputers*, MS thesis, Naval Postgraduate School, Monterey, California, September 1986, Chapter V "Conclusions and Future Research" pp. 73-81.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Dr. Harriet B. Rigas Dept. of Elec. Eng. and Sys. Science Michigan State University East Lansing, MI 48824	2
4.	Department Chairman, Code 62 Department of Electrical Engineering Naval Postgraduate School Monterey, CA 93943	1
5.	Professor Larry Abbott Department of Electrical Engineering, Code 62At Naval Postgraduate School Monterey, CA 93943	1
6.	Estado Maior da Armada Esplanada dos Ministérios Bloco 3 Brasília, D.F., 70055, Brazil	1
7.	Diretoria de Ensino da Marinha Praça Barão de Ladário, S/N Rio de Janeiro, R.J., 20091, Brazil	1
8.	Diretoria de Armamento e Comunicações da Marinha Rua Primeiro de Março, 118 - 19º andar Rio de Janeiro, R.J., 20010, Brazil	2
9.	CF Ronaldo Fiuza de Castro Instituto de Pesquisas da Marinha Rua Ipiru, s/n Jardim Guanabara, Ilha do Governador Rio de Janeiro, R.J., 21931, Brazil	1
10.	LCDR Julio Cesar Lopes de Albuquerque Brazilian Naval Commission 4706, Wisconsin Avenue Nw. Washington D.C., 20016	3

Thesis
A33816 Albuquerque
c.1 An extension to the multilevel logic simulator for microcomputers.